NICSO 2006

PROCEEDINGS OF THE WORKSHOP ON NATURE INSPIRED COOPERATIVE STRATEGIES FOR OPTIMIZATION

EDITORS:

David Alejandro Pelta
Natalio Krasnogor

David Pelta
Assistant Professor
Models of Decision and Optimization Research Group
Dept. of Computer Science & Artificial Intelligence
University of Granada
18071 - Granada, Spain

Natalio Krasnogor
Lecturer
Automatic Scheduling, Optimisation and Planning Research Group
School of Comp. Sciences and IT
University of Nottingham
Nottingham, NG81BB United Kingdom

Cover and Poster Design: Alejandro Sancho Royo
ISBN: 84-689-9323-9
Legal Deposit: DLGR 1332-2006
Printing: Rosillo's S.L.
Printed in Spain

Preface

Biological and natural processes have been influencing the methodologies in science and technology since a long time ago. The work of Wiener in cybernetics was influenced by feedback control processes observable in biological systems; McCulloch and Pitts' description of the artificial neuron was seeded from mathematical biology and electronics; the idea of "survival of the fittest" inspired the whole field of genetic algorithms and nowadays, the natural immune system is being considered as a source of inspiration for networks security.

From bacteria to humans, biological entities often engage on a rich repertoire of social interaction that could range from altruistic cooperation through open conflict. A paradigmatic case of social interaction is "cooperative problem solving", where a group of autonomous entities work together to achieve a certain goal. Examples of Nature-inspired coordination/cooperation algorithms may be found in: Ants Colonies, Artificial immune systems, Amorphous computing, Evolvable systems, Membrane computing, Quorum computing, Self assembly, Swarm intelligence, etc.

The purpose of the Workshop on Nature-Inspired Cooperative Strategies for Optimization (NICSO) is to extend investigations into emerging new areas inspired by nature, both at biological (i.e. micro) and behavioral (i.e. macro) levels for visionary concepts of information processing and architectures that, in turn, foster the role of cooperation/collaboration mechanisms in the context of problem solving.

NICSO was organized as an activity of the *Task Force on Nature-inspired Cooperative Strategies for Optimization*, supported by the 6th EU Framework Programme IST, under Project N^o 013569: "Nature-Inspired Smart Information Systems" `www.nisis.de`.

Additional support was granted by the Models of Decision and Optimization (MODO) Research group, Univ. of Granada, Spain, the Automated Scheduling, Optimisation and Planning (ASAP) research group, Univ. of Nottingham, UK, the Spanish Ministry of Science and Education through project TIN2005-08404-C04-01, the Andalusian Government through project MINAS TIC-00129-PE and the University of Granada.

As Workshop Chairs we are indebted to several people and institutions. We wish to sincerely acknowledge the work done by the members of the international programme committee in reviewing the submitted contributions.

We thank the ETSI Ingeniería Informática y de Telecomunicaciones for lending us the school' facilities, the members of MODO for the support with the organizational tasks, and we acknowledge the work of Alejandro Sancho in the proceedings' cover and poster design.

We thank also the plenary speakers Eshel Ben Jacob, Guy Theraulaz, Enrique Alba, David Anguita and Xiao-Zhi Gao for accepting our invitation. Their participation enhanced the quality of the Workshop.

Last but not least, we wish to sincerely thank José L. Verdegay for his constant advice and support.

David A. Pelta
Natalio Krasnogor
Granada, Nottingham June 2006

International Programme Committee

Uwe Aickelin , University of Nottingham, UK
Enrique Alba Torres , University of Malaga, Spain
Davide Anguita , Universita degli Studi di Genova, Italy
Cecilio Angulo , Technical University of Catalunya, Spain
Jaume Bacardit , University of Nottingham, UK
Mark Bedau , Reed College, USA
Larry Bull , University of the West of England, UK
José Manuel Cadenas , University of Murcia, Spain
Pierre Collet , Université du Littoral Côte d'Opale, France
Emilio Corchado , University of Burgos, Spain
Carlos Cruz Corona, University of Granada, Spain
Carlos Cotta , University of Malaga, Spain
Vincenzo Cutello , University of Catania, Italy
Xiao-Zhi Gao , Helsinki University of Technology, Finland
Jon Garibaldi , University of Nottingham, UK
Marian Gheorghe , University of Sheffield , UK
Jean-Louis Giavitto , Université d'Evry, France
Juan Ramón González, University of Granada, Spain
Steve Gustafson, GE Research, USA
Francisco Herrera , University of Granada, Spain
Natalio Krasnogor , University of Nottingham, UK
Derek Linkens , University of Sheffield, UK
Evelyne Lutton , INRIA, France
Vittorio Maniezzo , University of Bologna, Italy
Juan José Merelo , University of Granada, Spain
Belen Melian , University of La Laguna, Spain
José A. Moreno , University of La Laguna, Spain
Marcos Moreno , University of La Laguna, Spain
Andreas Nuernberger , University of Magdeburg, Germany
Gabriela Ochoa , University Simon Bolivar, Venezuela
Gheorghe Paun , University of Seville, Spain
David A. Pelta , University of Granada, Spain
Stefano Pizzuti , Energy New Technologies and Enrivonment Agency ENEA, Italy
Vitorino Ramos , Technical University of Lisbon, Portugal
Alejandro Sancho , Working group on Models of Decision and Optimization, Spain
Peter Siepmann, University of Nottingham, UK
Jim Smith , University of the West of England, UK
German Terrazas, University of Nottingham, UK
Jon Timmis , University of York, UK
José Luis Verdegay , University of Granada, Spain
Ronald Westra , University of Maastricht-Limburg, Netherlands

Table of Contents

Invited Talks

Peer Reviewed Contributions

Plenary Talks

The biological principles of swarm intelligence

Guy Theraulaz

CNRS UMR 5169
Centre de Recherches sur la Cognition Animale
Universit Paul Sabatier, 118, route de Narbonne
31062 Toulouse Cdex 4, France
theraula@cict.fr

The ecological success of social insects in general, and of many species of ants in particular, has been the starting point of a lot of ethological and theoretical studies. The focus of these studies has been, for instance, the organization of collective foraging activities, the division of labor inside a colony, the formation of complex tunneling networks, the clustering of brood items and corpses, or the collective building processes.

Experimental and theoretical investigations have suggested that self-organization, that is, the spontaneous emergence of a global order from local interactions and some degree of randomness at the individual level constitutes a plausible explanation to a wide range of collective phenomena without reference to any kind of central control or regulation: social insect colonies are collectively intelligent entities, the constituent units of which may be very simple and lack the sort of intelligence that emerges at the global level.

Many aspects of the "swarm intelligence" of social insects can be understood in terms of non-linear interactions between relatively simple individuals. More precisely, stigmergy and self-organization are the right concepts to describe a lot of instances of collective behaviors. These last ten years , numerous models of collective behaviors in social insects have provided powerful tools to solve engineering problems. These artificial swarm intelligent systems have been applied to many combinatorial optimization problems and dynamical optimization problems and they proved to be very efficient.

Here, I will review some of the basic coordination mechanisms used by social insects to collectively take a decision or build complex nests or networks architectures. Particular attention will be given to the way information is processed by a colony and how robust and efficient designs emerge as a consequence of self-organized activities.

References

1. Bonabeau, E. and Theraulaz, G. 2000. Swarm Smarts. Scientific American 282, 72-79.
2. Bonabeau, E., Dorigo, M. and Theraulaz, G. 1999. Swarm Intelligence: From Natural To Artificial Systems. Oxford University Press.
3. Bonabeau, E., Dorigo, M. and Theraulaz, G. 2000. The Social Insect Paradigm For Optimization And Control. Nature, 406, 39-42.
4. Buhl, J., Gautrais, J., Sole, R.V., Kuntz, P., Valverde, S., Deneubourg, J.L. and Theraulaz, G. 2004. Efficiency And Robustness In Ant Networks Of Galleries. The European Physical Journal B - Condensed Matter, 42: 123-129.

5. Buhl, J., Deneubourg, J.L., Grimal, A. and Theraulaz, G. 2005. Self-Organized Digging Activity In Ant Colonies. Behavioral Ecology And Sociobiology, 58: 9-17.
6. Camazine, S., Deneubourg, J.-L., Franks, N., Sneyd, J., Theraulaz, G. and Bonabeau, E. 2001. Self-Organization In Biological Systems. Princeton University Press.
7. Garnier, S., Jost, C., Jeanson, R., Gautrais, J., Asadpour, M., Caprari, G. and Theraulaz, G. 2005. Aggregation Behaviour As A Source Of Collective Decision In A Group Of Cockroach-Like-Robots, Lectures Notes In Computer Science, 3630 : 169-178.
8. Jeanson, R., Rivault, C., Deneubourg, J.L., Blanco, S., Fournier, R., Jost, C. and Theraulaz, G. 2005. Self-Organized Aggregation In Cockroaches, Animal Behaviour, 69: 169-180.
9. Theraulaz, G. and Bonabeau, E. 1995. Coordination In Distributed Building. Science, 269: 686-688.
10. Theraulaz, G. and Bonabeau, E. 1999. A Brief History Of Stigmergy. Artificial Life, 5: 97-116.
11. Theraulaz, G., Bonabeau, E., Nicolis, S., Sole, R.V., Fourcassie, V., Blanco, S., Fournier, R., Joly, J.L., Fernandez, P., Grimal, A., Dalle, P., and Deneubourg, J.L. 2002. Spatial Patterns In Ant Colonies. Proceedings Of The National Academy Of Sciences Usa, 99: 9645-9649.

Learning from Bacteria about Self-Organization

Eshel Ben Jacob

School of Physics and Astronomy
Tel Aviv University
69978nTel Aviv, Israel
eshel@tamar.tau.ac.il

A typical bacterial colony consists of numbers of bacteria exceeding the population of people on earth. I will present some of the remarkable patterns formed during colonial self-organization when the bacteria are exposed to versatile growth conditions mimicking the ones they are likely to encounter in Nature. I will then explain some of the simple yet ingenious methods invented by the bacteria for developing the incredible diversity of observed complex patterns.

It is now understood that bacteria can utilize intricate communication capabilities(e.g. quorum-sensing, chemotactic signaling and plasmid exchange) to cooperatively form (self-organize) complex colonies with elevated adaptability - the colonial pattern is collectively engineered according to the encountered environmental conditions. Bacteria do not genetically store all the information required for creating all possible patterns. Instead, additional information is cooperatively generated as required for the colonial self-organization to proceed.

In other words, the colony architecture is not created by a blue-print design according to a pre-stored detailed plan, but instead through a process of biotic self-organization. The elements (bacteria) store the information for creating the needed "tools" and the guiding principles needed for the colonial self-organization. Additional information is cooperatively generated as the organization proceeds following external stimulations. The outcome is an adaptable complex system that can perform many tasks, learn and change itself accordingly.

We can learn from bacteria self-organization to develop a new approach to construct man-made systems too complex for design. The idea is to let many collections of elements self-organize in a pre-engineered environment with which they can exchange information.

References

1. Ben-Jacob, E. (2003) Bacterial self-organization: co-enhancement of complexification and adaptability in a dynamic environment. Phil. Trans. R. Soc. Lond. A361, 1283-1312,
2. Ben Jacob, E., Shapira, Y. and Tauber, A.I. (2006) Seeking the foundations of cognition in bacteria: From Schrdinger's negative entropy to latent information Physica A vol. 359 ; 495-524
3. Ben-Jacob, E. et al. (1994) Generic modeling of cooperative growth patterns in bacterial colonies. Nature 368, 46-49

4. Ben Jacob, E. et al (2004) Bacterial Linguistic Communication and Social Intelligence Trends in Microbiology 12 (8) 366-372
5. Ben Jacob, E., Aharonov, Y. and Shapira, Y. (2005) Bacteria harnessing complexity Biofilms
6. Ben Jacob, E. and Levine, H. (2006) Self engineering capabilities of bacteria Interface.Published online. Rsif
7. Ben-Jacob, E. (1998) Bacterial wisdom, Godel's theorem and creative genomic webs. Physica A 248, 57-76
8. Ben Jacob, E. and Shapira, Y. (2004) Meaning-Based Natural Intelligence vs. Information-Based Artificial Intelligence. The Cradle of Creativity Edited by H. Nen Nun, Saarei Tzedk Jerusalem

Artificial Immune Systems and Their Applications

Xiao-Zhi Gao

Institute of Intelligent Power Electronics
Helsinki University of Technology
Otakaari 5 A, FI-02150 Espoo, Finland
http://powerelectronics.tkk.fi/
gao@cc.hut.fi

As we know, natural immune systems are complex and enormous self-defense systems with the remarkable capabilities of learning, memory, and adaptation [1]. Artificial Immune Systems (AIS), inspired by the natural immune systems, are an emerging kind of soft computing methods [2]. With the distinguishing features of pattern recognition, anomaly detection, data analysis, and machine learning, the AIS have recently gained considerable research interest from different communities [3]. Their successful industry applications include anomaly detection, optimization, fault diagnosis, pattern recognition, etc. [4].

This plenary talk focuses on the recent advances in the theory and applications of the AIS study, which consist of three main parts, as shown in Fig. 1.

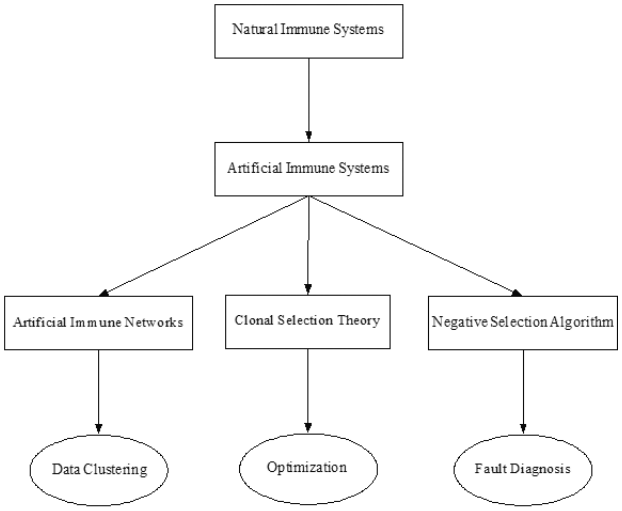


Fig. 1. Plenary talk on artificial immune systems and their applications

In the first part, we present the basic principles of the natural immune systems. A brief overview of the AIS is next given in the second part. The third part discusses three typical artificial immune systems, i.e., artificial immune networks, Clonal Selection Theory (CST), and Negative Selection Algorithm (NSA). Several representative artificial immune networks

models, such as the aiNet and RLAIS, together with their applications in data clustering are investigated. The fundamentals of the CST for engineering optimization will be explained as well [5]. We are also going to explore the essential principles of the NSA, and further present how to employ it in the area of fault diagnosis. Moreover, a few novel NSA-based fault diagnosis approaches are demonstrated [6].

Finally, we point out some future research topics of the AIS with applications. The promising fusion of the AIS and other soft computing methods, e.g., neural networks, fuzzy logic, and genetic algorithms, is especially emphasized.

References

1. C. A. Janeway, P. Travers, M. Walport, and M. Shlomchik, Immunobiology: The Immune System in Health and Disease, (5th ed.). New York, NY: Garland Publishing, 2001.
2. L. N. de Castro and J. Timmis, Artificial Immune Systems: A New Computational Intelligence Approach, London, UK: Springer-Verlag, 2002.
3. D. Dasgupta and N. Attoh-Okine, "Immunity-based systems: A survey" in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Orlando, FL, October 1997, pp. 369-374.
4. D. Dasgupta, Z. Ji, and F. González, "Artificial Immune System (AIS) research in the last five years", in Proceedings of the International Congress on Evolutionary Computation, Canbara, Australia, December 2003, pp. 123-130.
5. X. Wang, X. Z. Gao, and S. J. Ovaska, "Artificial immune optimization methods and applications - A survey", in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, The Hague, The Netherlands, October 2004, pp. 3415-3420.
6. X. Z. Gao, S. J. Ovaska, X. Wang, and M.-Y. Chow, "A neural networks-based adaptive negative selection algorithm with application in motor fault diagnosis" in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, The Hague, The Netherlands, October 2004, pp. 3408-3414.

Parallelizing Nature Inspired Optimization Methods

Enrique Alba Torres

GISUM group (NEO line)
University of Málaga
Málaga, Spain
eat@lcc.uma.es

Optimization, search, and learning are healthy field targeted to find efficient solutions and algorithms to solve problems either in the academia and the industry. As the solved problems became harder a need for very efficient tools arose in computer science. One way to deal with this new requested efficiency is to use several computers to solve the same problem, either by using shared memory multiprocessors, clusters of machines or the Internet itself (also in grid). This talk will include some introduction to the field of parallelizing nature inspired methods and will develop on important issues like exact versus heuristic algorithms, software tools, redefinition of metrics, and a set of illustrating examples to give the audience a brief state of the art in parallelism and optimization algorithms.

The talk will start by offering a wide definition of optimization through the utilization of nature inspired algorithms like genetic algorithms, simulated annealing, ant colony systems and some other methods using natural metaphors to solve complex problems. In addition to pointing out the difficulties and advantages of dealing with parallel techniques we will go through different facts occurring in the field of parallelization at least including the following ones:

1. Model and implementation are different
2. Metrics need a revision
3. Superlinear speedup is a fact
4. Heterogeneity is a must nowadays
5. The experimental setup is important
6. Algorithms are software
7. Other facts

In every case we will include examples illustrating the different milestones that researchers must face when working in parallel algorithms. Relevant books and projects will be briefly outlined, and a final discussion on applications will complete the presentation. Among the applications, combinatorial optimization, numerical continuous problems, telecoms, bioinformatics and neural networks design will reveal the true challenges open to researchers, and how parallel techniques can really endorse the creation of new techniques able of unseen efficiency and accuracy.

NiSIS: A European Concerted Action for Nature-inspired Smart Information Systems

Davide Anguita

DIBE - Dept. of Biophysical and Electronic Eng.
University of Genoa
Via Opera Pia 11A, 16145, Genoa, Italy
Davide.Anguita@unige.it

In recent years, biological and natural processes have been influencing the methodologies in science and technology in an increasing manner. Thus, the work of Wiener in cybernetics was heavily influenced by feedback control processes observable in biological systems. McCulloch and Pitts description of the artificial neuron was seeded from mathematical biology and electronics. Watson and Cricks description of the DNA molecule and the subsequent immense strides now being made in genomics offer similar possibilities for advances. Also, there is a deepening knowledge of natural immunological systems which are able to recognise and eliminate foreign bodies, thus providing an obvious paradigm for vital future security methodologies for advanced information networks.

NiSIS is a European Project under the Co-ordinated Action (CA) scheme with the following overall mission aims:

to co-ordinate multi-disciplinary studies and research endeavours into the development and utilisation of intelligent paradigms in advanced information systems design.

Information systems manipulate and transmit/receive data in such a manner as to convey information and knowledge between humans and machines. They have across-scale magnitudes of components and complexity in size and behaviour. Intelligent (Information) Systems must be able to learn and adapt on-line by either updating parameters or structures of the system in a stable and reliable manner. To extend investigations into emerging new areas inspired by nature, both at biological (i.e. micro) and behavioural (i.e. macro) levels for visionary concepts of information processing and architectures. Under this mission statement, the main objectives of NiSIS are to:

- Encourage cross-disciplinary team-based thinking to cross-fertilise engineering and life science understanding into advanced inter-operable systems.
- Progress the theme of adaptivity beyond curiosity and basic earlier engineering concepts and theory, via the spur of naturally-occurring phenomena and self-emergent systems.
- Elaborate the themes of hierarchy, modularity, redundancy, learning capacity etc in pursuit of greater robustness and reliability against uncertainties, time-variations and fault conditions for large information systems.
- Incorporate the large body of knowledge on systems dynamics, modelling and identification/estimation into hybrid structures based on intelligent paradigms.
- Develop a Taxonomy and Strategic Roadmap to describe the state-of-the-art knowledge in the different disciplines about intelligent systems and to signpost a future for inte-

gration of deeper nature-inspired concepts into smart devices and systems. This aims to prepare for long term goals leading beyond FP6.
- Provide training and education across the relevant disciplines via workshops, symposia, Best Practice Guidelines etc.
- Foster Technology Transfer via competitive team-based feasibility Workshops on realistic benchmarking problems.
- Co-ordinate the Project with a management structure which fosters cross-disciplinary endeavours via Focus Groups, Task Forces on innovative concepts, Annual symposia etc.
- Grow a new set of teams/groups via recruitment of new partners/partners steadily over 3 years, with a target of about 100 actively participating members.
- Encourage promising young researchers in the field, spin-off ventures and SME via interchange of students and researchers using Web-based material and short Schools providing expertise transfer.
- Establish links with other scientific projects and organisations, such as EU Networks of Excellence, which are working in the generic field of intelligent systems, via Web-site data and information

At the core of the project is a set of three major research areas via which the partners deliver the scientific and technological contributions to the programme. The three research areas (called Focus Groups) are:

- NiDT: Nature-inspired Data Technologies
- NiN: Nature-inspired Networks
- NIMOC: Nature-inspired Systems Modelling, Optimization and Control

In addition to the three Focus Groups a TTE (Technology transfer, Training and Education) Group gives added-value in terms of cross-disciplinary instruction and motivation for an upcoming generation of young researchers as well as for the dissemination and standarization. It also provides the necessary technology transfer, training and education in order to promote the area and enlarge the domain of potential users. The overall integration of the project results is overseen by an ITB (Integrated Technology Board).

Every European institution with goals and activities which are related to NiSIS can apply to become a Partner. All Partners are represented by contact persons and by participating researchers. An applying Partner should complete a questionnaire related to its competencies in nature-inspired systems research, a description of the institution as well as a summary of the work performed in the area of NiSIS and its choice and motivation to join an NiSIS group. Partner applications are approved by the Steering Committee taking into consideration the advice of the relevant group.

Everybody interested in NiSIS activities may register on the mailing list. More information is available on the project web site: `www.nisis.de`

Peer Reviewed Contributions

Natural Behavior Based Metaheuristics for the Vehicle Routing Problem with Time Window

***Rodrigo A. Garrido*[1] *and Zdenko Koscina*[1]**

Department of Transport Engineering and Logistics
Pontificia Universidad Católica de Chile
Santiago, Chile
rgarrido@ing.puc.cl

Abstract. *This paper presents a comparison between two metaheuristics based on Adaptive Memory: Ant Colony Optimization and Tabu Search with Adaptive Memory. These metaheuristics are especially adequate for solving network optimization problems where a shortest path search is a critical step. The metaheuristics were tested on a set of benchmark instances of the Vehicle Routing Problem with Time Windows – a well known NP-complete problem. Both metaheuristics found solutions competitive with the best solutions published in the literature for the studied instances. The numerical results showed no dominance (in a Pareto sense) of either metaheuristics. While Ant Colony Optimization method was more efficient at minimizing the number of vehicles, the Tabu Search with Adaptive Memory was better at minimizing the total traveled distance. The results suggest that a hybrid method that captures the best of each metaheuristic could be successful in the solution of these type of network optimization problems.*

1 Introduction

One of the most appealing problems in combinatorial optimization and its consequent application in logistics is the Vehicle Routing Problem (VRP), first formulated in the 50's by [5], where a homogeneous fleet of vehicles, based on e central depot, serves a set of geographically dispersed customers, each one with a demand for a certain commodity. The VRP is that of finding the set of routes that starts and end at the depot and satisfy all the customers' demands at minimum cost. The solution must obey constraints such as vehicles autonomy and capacity. This problem is known to be NP-complete, i.e., it has not yet been found an efficient algorithm to solve it within "reasonable" computation time for general cases. One of the most interesting extensions to the VRP is its time-constrained version called Vehicle Routing Problem with Time Windows (VRPTW). In this extension, each customer specifies earliest and latest times at which the vehicles can provide service. This problem, being a generalization of the VRP it is also an NP-complete problem, and most real-world applications cannot be solved optimally.

From the solution viewpoint, the VRPTW is a multiobjective problem. Typical objectives to optimize are: the number of vehicles to use, total traveled distance, total routing time and waiting time. The most common approach to its solution has traditionally been through a hierarchical structure, usually with the number of vehicles at the top hierarchy,

followed by the traveled distance.

Two types of approaches have been used to solve the VRPTW: exact optimizing methods and approximation heuristics. The former has been the preferred approach by theoreticians whereas the latter has been the choice of applied scientists. Most of the real-world applications of the VRPTW (e.g. in urban logistics or military deployments) have a size that precludes the use of optimizing methods. Consequently, there has been a raising interest in developing heuristics that can find a good solution in a reasonable amount of time, even though it is not necessarily the optimum.

In the recent years, a new breed of general scope heuristics (metaheuristics) based on natural behavior has emerged. These metaheuristics try to copy some of the successful strategies followed by animals in search for food. Some of these approaches can be categorized as adaptive memory programming. This umbrella broadly covers metaheuristics, such as Tabu Search, Scatter Search, Genetic Algorithms, Ant Colonies among others [18], when their search power is improved by taking advantage of a learning process in which memory plays a vital role. These metaheuristics share the following three characteristics (also found in the animal kingdom): they memorize characteristics of the generated solutions during the search process, they create initial solutions from memory stored information, and third, they apply a local search to improve the initial solution.

This paper presents a comparative study of two of the most promising AMP metaheuristics: Tabu Search with Adaptive Memory (TSAM) and Ant Colony Optimization (ACO). The ACO method [9], [10] is inspired by the behavior of ants in finding paths from the colony to food. The TSAM is an advance form of TS that embodies a framework that includes long term memory with associated intensification and diversification strategies thus exploiting a collection of strategic memory components. The thrust that gives these methods their distinctive character is the systematic use of adaptive memory — in contrast to memoryless approaches (e.g. simulated annealing).

The comparison is made on the basis of their performance on a set of standard test problems with known solution and their qualitative ability to solve real-world instances.

2 Solution Approaches

Due to the complexity of the VRPTW, most of the solution approaches have been based on heuristic methods. These heuristics can be grouped into three categories: construction heuristics [2], local search [4] and metaheuristics [3]. There are however, other heuristics that do not belong to these broad categories but have received much less attention from the specialized literature, such as the optimization based heuristics [13] and asymptotically optimal heuristics [1]. A metaheuristic is a heuristic method for solving a general class of computational problems by combining user designed procedures - usually heuristics- efficiently.

One disadvantage of some traditional heuristics and metaheuristics is their inability to escape from local optima. This feature usually leads to poor solutions because the solver

gets "trapped" in a local optimum and no further search is done in different regions where a better solution may lay. For this reason several extensions to these heuristics have emerged. These extensions allow intensification and diversification strategies to both explore good solutions in a neighborhood of an available solution and elsewhere. These new (or improved) metaheuristics are in the frontier of operations research and artificial intelligence. Unfortunately there is no general dominance among the various available metaheuristics to solve NP-complete problems, even within sets of similar instances [15]. Therefore we chose to implement two of the most promising alternatives for the problem at hand: the Tabu Search with Adaptive Memory (TSAM) and Ant Colony Optimization (ACO) which are described below. The TSAM performance on vehicle routing with soft time windows was proven successful by [17]; the ACO showed excellent performance in dynamic vehicle routing problems [14].

3 A natural approach to the search for shortest paths

3.1 The Ant Colony Optimization Heuristic

The attempt to develop algorithms inspired by behavioral aspects of natural species has led to study the ants' behavior in search for shortest paths. That is the scope of the so called ant colony optimization method (ACO), the most successful and widely recognized algorithmic technique based on natural species behavior [8]. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around a graph representing the problem to solve.

ACO algorithms have been used to produce near-optimal solutions to the traveling salesman problem. They have an advantage over simulated annealing and genetic algorithm approaches when the graph may change dynamically; the ant colony algorithm can be run continuously and adapt to changes in real time. This is of interest in network routing and urban transportation systems. ACO uses several agents to scout the solution space in search for productive regions.

There are many implementations of ACO algorithms. We implemented the version called Ant Colony System (ACS) put forward by [6] as an improvement to the original version by [7]. The iterative procedure is as follows: at each iteration m ants build a tour moving from one point in space to another. When an ant k finds a node, it randomly selects the next node to visit, favoring arcs with a higher concentration of pheromone. When the ant k is located at node i, it chooses the next node j to visit in a pseudo-random manner: with probability q the node with the highest value of $\tau_{ij} \cdot [\eta_{ij}]^{\beta}$will be selected, and with probability (1-q) the node j is chosen according with the following function:

$$p_{ij}^{k} = \frac{\tau_{ij} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} \tau_{il} \cdot [\eta_{il}]^{\beta}} \tag{1}$$

where N_i^k represents the set of nodes adjacent to node i not yet visited by the ant k; $\tau_{ij}$is the amount of pheromone in arc (i, j); η_{ij} is the inverse of the distance of arc (i, j), and β is a

control parameter. The parameter q controls the degree of intensification and diversification.

Only the ant who found the best tour is allowed to lay pheromone on the traveled arcs as follows:

$$\Delta\tau_{ij}^{=} \begin{cases} 1/L \; if \; (i,j) \in T \\ 0 \;\; \sim \end{cases} \tag{2}$$

where T represents the best tour found and L is the length of that tour.

The pheromone is evaporated on-line, i.e., each time an ant selects an arc, a local evaporation takes place according to the following expression:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{3}$$

where ρ is a control parameter such that $0 \leq \rho \leq 1$. This local evaporation decreases the concentration at arc (i, j) (note that it could also increase it if τ_0 is big enough). A decreasing concentration on a path also decreases its probability of being chosen by other ants in subsequent iterations.

[9] extend the ACO metaheuristics considering two ant colonies working in parallel exchanging information with a central system that directs the search. This heuristic is called Multiple Ants colony System (MACS). One colony works in minimizing the total traveled distance, while the other one attempts to minimize the number of routes.

Computationally, there is a cost to be paid for using two colonies instead of one, but in all the cases the extra cost is rewarded with better solutions with a moderate increase in the execution time. We performed a comparison between implementations with one and two colonies. As an example, Table 3.1 shows a comparison between an implementation with one colony that minimizes both the number of vehicles and the traveled distance, against a two-colonies implementation where one colony minimizes the number of vehicles while the other minimizes the traveled distance, on the Solomon's groups of instances R1 and R2 with 100 customers (see section 4.1 for a description). Although the time required for finding the solution with two colonies is about twice that of the single-colony implementation, the absolute values of the execution time are rather small but the solution improves significantly. Consequently, the ACO tested in this paper is the MACS.

Table 1. Comparison of ACO heuristics with one and two colonies on the Solomon's groups R1 and R2.

Implementation	Instance	Vehicles	Distance	Comp. time (min)
1 Colony	R1	14.42	1573.39	0.97
	R2	3.64	1539.57	1.32
2 Colonies	R1	13.17	1368.45	2.46
	R2	3.09	1294.27	2.64

3.2 Tabu Search with Adaptive Memory

The word *tabu* comes from Tongan (the language of aborigines of an island in Polynesia); according to Captain James Cook, things that are *forbidden to be eaten, or made use of, they say, that it is taboo* (Marra, 1777).

Tabu Search (TS) methods employ responsive exploration, which integrates the basic principles of intelligent search by exploiting good solution features, while at the same time exploring new promising regions. The TS approach implemented in this research is that of [17], who combined the standard methods of TS with a powerful technique of probabilistic intensification/diversification that falls within the umbrella of adaptive memory programming [18]. The original approach was designed for solving the VRP with Soft Time Windows and hence, this section described an adaptation of it to our problem with hard time windows. Adaptive memory (AM) is based on principles of reinforced learning. AM is implemented here as a list that contains a set of routes extracted from the best solutions found at any stage during the TS algorithm. Its purpose is to provide new initial solutions for the TS. All the routes belonging to a given solution are stored sequentially in the AM list, according to the value of their objective function. Thus, the best solution routes are stored in the first positions of the memory. The AM list is initialized with a set of p feasible solutions. At each iteration, a new solution is assembled for the TS by combining the routes in the AM list. At the end of the TS the AM list is updated. The effect of the AM on the TS performance has been shown to be successful on the Solomon's instances [12].

4 Computational Results

4.1 Benchmark Instances

The two metaheuristics described above (MACS and TSAM) were tested on two groups of instances:

- Medium Size: the 56 Solomon's instances with 100 customers [16]. These instances cover six types of problems according to the characteristics of their constraints and the spatial arrangements of the customers. Thus, there are instances with tight time window and capacity constraints (groups C1, RC1 and R1), as well as those with more slackness (groups C2, RC2 and R2). Groups C1 and C2 have customers arranged in clusters over the region, whereas groups R1 and R2 present customers uniformly distributed over the region. Groups RC1 and RC2 present a mixture of these spatial arrangements.
- Large Size: Homberger's instantes with 200, 400, 600, 800 and 1000 clients [11]. The nomenclature is analogue to that of the Solomon's instances.

4.2 Results for the MACS heuristic

The parameters used for the MACS implementation were m= 10; q=0.9; β= 2.0 y ρ= 0.1. The experiments were executed on a single processor Pentium IV 2,400 Mhz. Each run consisted of 400 iterations. Table 2 shows the average results for the Solomon's instances. The first row indicates the type of problem and in parenthesis the number of instances belonging to that group on which the average was computed.

Table 2. Average results for the MACS heuristic on the Solomon instances

	Best reported solution		MACS		
Group	Vehicles	Distance	Vehicles	Distance	Comp. Time (min.)
C1(9)	10.00	828.38	10.00	891.38	2.00
C2(8)	3.00	589.86	3.00	677.09	1.94
R1(12)	11.92	1209.89	13.17	1368.45	2.46
R2(11)	2.73	951.91	3.09	1294.27	2.64
RC1(8)	11.50	1384.16	12.50	1505.72	4.70
RC2(8)	3.25	1119.35	3.38	1402.12	4.87

The MACS heuristic presents an acceptable performance on the 100-customers Solomon's instances. The average deviation from the best published solutions is 7,83 % for the number of vehicles and 18,51 % for the traveled distance. In four of the 56 instances (C101, C105, C107 y C201), the MACS heuristic matched the best solution reported in the literature. Table 2 also shows that the best performance is reached in problems of type 1 (C1, R1 y RC1) where the time windows and capacity constraints are tight. The performance of this heuristics looses power when there is abundant slack in the constraints. In addition, when the customers are clustered together the MACS heuristic yields better results than when the customers are randomly scattered throughout the region. The average computation time over the whole range of instances is about 3 min., which is fairly good for a single processor with moderate speed (2.4 GHz).

Table 3. Average results for the MACS heuristic on the Homberger instances

	Best reported solution		MACS		
Group	Vehicles	Distance	Vehicles	Distance	Comp. Time (min.)
R1-2-1	19	5,024.65	21	6741.44	10.13
R1-4-1	38	11,084.00	40	37900.95	68.17
R1-6-1	59	21,131.09	61	40537.08	219.46
R1-8-1	79	39,612.20	81	72550.00	507.71
R1-10-1	100	54,145.31	101	110122.61	989.58
R2-2-1	4	4,501.80	5	5150.21	10.83
R2-4-1	8	9,257.92	10	13910.92	68.17
R2-6-1	11	18,325.60	13	35050.40	199.63
R2-8-1	15	28,440.28	17	163186.13	444.88
R2-10-1	19	42,922.56	21	90788.22	855.04

Table 3 shows the results for the MACS heuristic on the large-size group of instances (Homberger's instances). The MACS performance is rather poor when compared to the best published solutions. For example, for the case R-2-8-1, the found solution is five times bigger than its published counterpart. The average deviation with respect to the best solutions is 11,5% in the number of vehicles and more than twice for the traveled distance. Not only the objective functions are poor but also the computation times are considerably large. In fact, beyond 400 clients, the MACS heuristic is no longer viable in practice. For example,

for the instance R110-1 (1,000 clients) it takes more than 16 hours of CPU in a Pentium IV 2.4GHz).

4.3 Results for the TSAM heuristic

Each instance was ran once with the following parameters: initial solutions = 20; AM's stopping criterion: 100 updates; AM's size: 30 solutions; TS stopping

criterion: 50 iterations; tabu list: 100.000 routes; tabu tenure 25. These parameters were found to be the most efficient for the type of instances to be solved [12]. Table 4 shows the average results for each Solomon's instances group. The first row indicates the type of problem and in parenthesis the number of instances belonging to that group on which the average was computed.

Table 4. Average results for the TSAM heuristic on the Solomon instances

	Best reported solution		TSAM		
Group	Vehicles	Distance	Vehicles	Distance	Comp. Time (min.)
C1(9)	10.00	828.38	10.00	841.34	0.12
C2(8)	3.00	589.86	3.00	594.61	0.68
R1(12)	11.92	1209.89	12.92	1214.18	0.10
R2(11)	2.73	951.91	3.27	993.51	1.75
RC1(8)	11.50	1384.16	13.00	1385.39	0.17
RC2(8)	3.25	1119.35	3.75	1134.75	2.22

The TSAM heuristic performs well when compared to the best solutions found in the literature with computation times reasonably low (less than three minutes in all the instances). The average deviation from the best published solutions is 9,7% for the number of vehicles and 1,99% for the traveled distance. The quality of the solutions (as compared to the best published solutions) decreases as the constraints become less stringent and the customer distribution becomes random. Indeed, for groups C1 and C2 the TSAM heuristic gives close results to those of the best reported solutions, whereas for the groups R2 and RC2 its performance is worse.

The difference between the best solution reported in the literature and the one found by the TSAM heuristic is more notorious for the Hamberger's instances (see Table 5). The latter is due to the size of the instances (which translates into a bigger solution's space). The difference is more pronounced in the instances of type R2, where the constraints are not tight. On the other hand, for the instances R1-2-1 and R2-2-1, the TSAM heuristic finds a solution where the traveled distance is actually smaller than the best published solution, but at the expense of a larger fleet. The Homberger's instances show more clearly the difference in computational effort when solving instances with tight constraints in contrast to those with more slack. For example, the instance R110-1 is solved in 3.7 min whereas the instance R210-1 was solved in 45.3 min (both with 1,000 clients).

For the mid-size instances there is no clear winner because there are trade-offs. However, the TSAM heuristic outperforms its MACS counterpart more often. In groups C1, C2

Table 5. Average results for the TSAM heuristic on the Homberger's instances

	Best reported solution		TSAM		
Group	Vehicles	Distance	Vehicles	Distance	Comp. Time (min.)
R1-2-1	19	5,024.65	22	4,781.56	0.22
R1-4-1	38	11,084.00	41	11,249.12	0.47
R1-6-1	59	21,131.09	61	26,809.18	1.63
R1-8-1	79	39,612.20	82	47,455.99	2.73
R1-10-1	100	54,145.31	101	75,515.78	3.67
R2-2-1	4	4,501.80	5	4,171.49	9.27
R2-4-1	8	9,257.92	9	10,008.81	18.73
R2-6-1	11	18,325.60	13	20,647.20	35.30
R2-8-1	15	28,440.28	18	33,627.88	47.38
R2-10-1	19	42,922.56	21	57,766.59	45.32

and R1 TSAM dominates MACS, whereas in groups R2, RC1 and RC2 there are trade-offs between both metaheuristics. Nevertheless, in practice, the most relevant cost is always the number of vehicles. Indeed, the direct cost associated to the equipment/fuel plus the depreciation cost is often the dominant term in fleet management accounting. The extent to which these two types of costs differ from each other cannot be analytically determined as is it certainly varies from case to case. If a hierarchical objective function were considered, where the first objective is the number of vehicles, the MACS metaheuristic would outperform the TSAM in the groups R2, RC1 and RC2. Even though the percentage of differences in computation times for the Solomon's instances seem vastly favorable to the TSAM metaheuristic, the absolute values are quite modest. In fact, less than three minutes of difference for the whole solution (100 customers) is not relevant from a practical viewpoint.

The performance of both metaheuristics on the Homberger's instances show more differences. In fact, this comparison suggests that the MACS implementation is not efficient for large instances. Indeed, the computation time and the quality of the results (see tables 3 and 5) are most favorable to the TSAM implementation.

From a practical standpoint, the best heuristic should also be flexible enough to be able to incorporate side constraints without adding high complexity to the procedure. This is the case in most real-world applications where the assumptions of the VRPTW are sometimes too rigid. One of the most common violations of the original assumptions is the use of a heterogeneous fleet. In this flank the MACS metaheuristic outperforms the TSAM due to the simplicity in the heuristics modification. In fact, a great variety of side constraints can be incorporated at a moderate cost in the MACS heuristic when the set of feasible customers is defined (N_i^k) during the construction phase *new active ant* for the k-eth ant at node i. The characteristics of the vehicle are incorporated each time the ant leaves a node to begin a new route. This inclusion is considerable more difficult in the TSAM metaheuristic adding more complexity to the search. Indeed, the local search in the TS scheme becomes considerably more time consuming.

5 Final Remarks and Future Research Lines

We tested the performance of two metaheuristics to solve the vehicle routing problem with time windows: the Tabu Search with Adaptive Memory (TSAM) and the multiple Ant Colony System (MACS). These metaheuristics copy aspects of natural behavior such as the search for food in the case of ant colonies and the use of short term/long term memory in the case of TSAM. The metaheuristics were applied to two sets of tests: Solomon's instances (mid-size) and Homberger's instances (large-size). Both metaheuristics performed well in most of the mid-size instances when compared in a multi-attribute fashion against the best published solutions. No absolute dominance between them was found. On one hand, the tabu search scheme finds lower traveled distances quicker than the ant colonies implementation; on the other hand the ant colonies implementation find a smaller number of vehicles and presents more flexibility to include side constraints usually found in real-world applications. However, for the large-size instances, the performance of the MACS implmentation is rather poor when compared with the TSAM method. We believe that a hybrid scheme that takes advantage of the features of both approaches could perform significantly better than the pure approaches followed in this paper. For instance, the advantage of MACS in clustered settings is remarkable: four instances (C101, C105, C107 y C201) found the same "best solution" reported in the literature with the most sophisticated algorithms. The TSAM's ability to find shorter routes in different regions within the area under study is also remarkable. These findings give hope for an optimal hybridation of both methods.

Acknowledgements

This research has been supported in part by the Chilean National Science and Technology Foundation (FONDECYT) under Grant 1050673 and the Maryland Transportation Initiative at the University of Maryland.

References

1. J. Bramel and D. SimchiLevi. Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows. *Operations Research*, 44(3):501–509, 1996.
2. I. Braysy and M. Gendreau. Vehicle routing problem with time windows, part 1: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.
3. I. Braysy and M. Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.
4. O. Braysy. Fast local searches for the vehicle routing problem with time windows. *Infor*, 40(4):319–330, 2002.
5. G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.
6. M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.
7. M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *Ieee Transactions on Systems Man and Cybernetics Part B-Cybernetics*, 26(1):29–41, 1996.
8. M. Dorigo and T. Stutzle. *Ant Colony Optimization*. 2004.

9. L. M. Gambardella, E. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.
10. L. M. Gambardella, E. Taillard, and Agazzi G. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In M. Dorigo D. Corne and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, London, UK, 1999.
11. J. Homberger and H. Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *Infor*, 37(3):297–318, 1999.
12. Z. I. KOSCINA. *El problema de ruteo vehicular con ventanas de tiempo: tres heurísticas de solución*. Master's thesis, Pontificia Universidad Catolica de Chile, 2005.
13. Y. A. Koskosidis, W. B. Powell, and M. M. Solomon. An optimization-based heuristic for vehicle-routing and scheduling with soft time window constraints. *Transportation Science*, 26(2):69–85, 1992.
14. R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.
15. O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stutzle. A comparison of the performance of different metaheuristics on the timetabling problem. *Practice and Theory of Automated Timetabling Iv*, 2740:329–351, 2003.
16. M. M. Solomon. Algorithms for the vehicle-routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.
17. E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.
18. E. D. Taillard, L. M. Gambardella, M. Gendreau, and J. Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16, 2001.

A Cellular Genetic Algorithm for Multiobjective Optimization

A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba

Departamento de Lenguajes y Ciencias de la Computación
E.T.S. Ingeniería Informática
Campus de Teatinos, 29071 Málaga (Spain)
antonio,durillo,flv,bernabe,eat@lcc.uma.es

Abstract. *This paper introduces a new cellular genetic algorithm for solving multiobjective continuous optimization problems. Our approach is characterized by using an external archive to store non-dominated solutions and a feedback mechanism in which solutions from this archive randomly replaces existing individuals in the population after each iteration. The result is a simple and elitist algorithm called MOCell. Our proposal has been evaluated with both constrained and unconstrained problems and compared against NSGA-II and SPEA2, two state-of-the-art evolutionary multiobjective optimizers. For the used benchmark, preliminary experiments indicate that MOCell obtains competitive results in terms of convergence, and it clearly outperforms the other two compared algorithms concerning the diversity of solutions along the Pareto front.*

1 Introduction

Most optimization problems in the real world involve the minimization and/or maximization of more than one function. Generally speaking, multiobjective optimization does not restrict to find a unique single solution of a given multiobjective optimization problem (MOP), but a set of solutions called *non-dominated solutions*. Each solution in this set is said to be a *Pareto optimum*, and when they are plotted in the objective space they are collectively known as the *Pareto front*. Obtaining the Pareto front of a given MOP is the main goal of multiobjective optimization. In general, the search spaces in MOPs use to be very large, and evaluating the functions can require a significant amount of time. These features make difficult to apply deterministic techniques and, therefore, stochastic techniques have been widely proposed within this domain. Among them, evolutionary algorithms (EAs) have been investigated by many researchers, and some of the most well-known algorithms for solving MOPs belong to this class (e.g. NSGA-II [10], PAES [12], and SPEA2 [17]).

EAs are especially well-suited for tackling MOPs because of their ability for finding multiple trade-off solutions in one single run. Well-accepted subclasses of EAs are Genetic Algorithms (GA), Genetic Programming (GP), Evolutionary Programming (EP), and Evolution Strategies (ES). These algorithms work over a set (*population*) of potential solutions (*individuals*) which undergoes stochastic operators in order to search for better solutions. Most EAs use a single population (panmixia) of individuals and apply the operators to them as a whole (see Fig. 1a). Conversely, there exist the so-called structured EAs, in which the population is decentralized somehow. Among the many types of structured EAs, *distributed* and *cellular* models are two popular optimization variants [4,6] (see Fig. 1b and Fig. 1c). In many cases, these decentralized algorithms provide a better sampling of the search space,

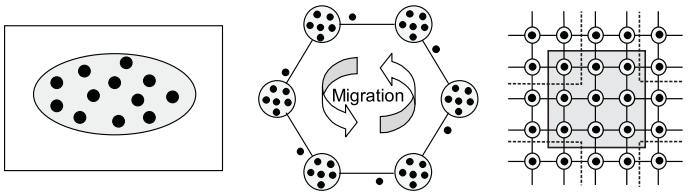


Fig. 1. Panmictic (a), distributed (b), and cellular (c) GAs

resulting in an improved numerical behavior with respect to an equivalent algorithm in panmixia.

In this work, we focus on the cellular model of GAs (cGAs). In cGAs, the concept of (small) *neighborhood* is intensively used; this means that an individual may only interact with its nearby neighbors in the breeding loop [14]. The overlapped small neighborhoods of cGAs help in exploring the search space because the induced slow diffusion of solutions through the population provides a kind of exploration (diversification), while exploitation (intensification) takes place inside each neighborhood by genetic operations. These cGAs were initially designed for working in massively parallel machines, although the model itself has been adopted also for mono-processor machines, with no relation to parallelism at all. Besides, the neighborhood is defined among tentative solutions in the algorithm, with no relation to the geographical neighborhood definition in the problem space.

cGAs have proven to be very effective for solving a diverse set of single objective optimization problems from both classical and real world settings [1, 2], but little attention has been paid to its use in the multiobjective optimization field. In [13], a multiobjective evolution strategy following a predator-prey model is presented. This is a model similar to a cGA, because solutions (preys) are placed on the vertices of an undirected connected graph, thus defining neighborhoods, where they are 'caught' by predators. Murata and Gen presented in [15] an algorithm in which, for an n-objective MOP, the population is structured in an n-dimensional weight space, and the location of individuals (called cells) depends on their weight vector. Thus, the information given by the weight vector of individuals is used for guiding the search. A metapopulation evolutionary algorithm (called MEA) is presented in [11]. This algorithm is a cellular model with the peculiarity that disasters can occasionally happen in the population, thus dying all the individuals located in the disaster area (extinction). Additionally, these empty areas can also be occupied by individuals (colonization). Thus, this model allows a flexible population size, combining the ideas of cellular and spatially distributed populations. Finally, Alba et al. proposed in [3] cMOGA, the unique cellular multiobjective algorithm based on the canonical cGA model before this work, to the best of our known. In that work, cMOGA was used for optimizing a broadcasting strategy specifically designed for mobile ad hoc networks.

Our proposal is called MOCell, and it is, like in the case of [3], an adaptation of a canonical cGA to the multiobjective field. MOCell uses an external archive to store the non-dominated solutions found during the execution of the algorithm, like many other multiobjective evolutionary algorithms do (e.g., PAES, SPEA2, or cMOGA). However, the main feature characterizing MOCell with respect to these algorithms is that a number of solutions are moved back into the population from the archive after each iteration, replacing randomly selected existing individuals. The contributions of our work can be summarized as follows:

- We propose a new cGA for solving continuous MOPs. The algorithm uses an external archive and a feedback of solutions from the archive to the population.
- The algorithm is evaluated using a benchmark of constrained and unconstrained MOPs.
- MOCell is compared against NSGA-II and SPEA2, two state-of-the-art GAs for solving MOPs.

The rest of the paper is organized as follows. In Section 2, we present several basic concepts on multiobjective optimization. In Section 3, we describe MOCell, our proposal for facing MOPs. Our results are presented and discussed in Section 4. Finally, in Section 5 we give our main conclusions and suggest some future research lines.

2 Multiobjective Optimization Fundamentals

In this section, we include some background on multiobjective optimization. In concrete, we define the concepts of MOP, Pareto optimality, Pareto dominance, Pareto optimal set, and Pareto front. In these definitions we are assuming, without loss of generality, the minimization of all the objectives. A general multiobjective optimization problem (MOP) can be formally defined as follows:

Definition 1 (MOP). Find a vector $\boldsymbol{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ which satisfies the m inequality constraints $g_i(\boldsymbol{x}) \geq 0, i = 1, 2, \ldots, m$, the p equality constraints $h_i(\boldsymbol{x}) = 0, i = 1, 2, \ldots, p$, and minimizes the vector function $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})]^T$, where $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables.

The set of all values satisfying the constraints defines the *feasible region* Ω and any point $\boldsymbol{x} \in \Omega$ is a *feasible solution*. As mentioned before, we seek for the *Pareto optima*. Its formal definition is provided next:

Definition 2 (Pareto Optimality). A point $\boldsymbol{x}^* \in \Omega$ is Pareto Optimal if for every $\boldsymbol{x} \in \Omega$ and $I = \{1, 2, \ldots, k\}$ either $\forall_{i \in I} (f_i(\boldsymbol{x}) = f_i(\boldsymbol{x}^*))$ or there is at least one $i \in I$ such that $f_i(\boldsymbol{x}) > f_i(\boldsymbol{x}^*)$.

This definition states that $\boldsymbol{x}^*$ is Pareto optimal if no feasible vector $\boldsymbol{x}$ exists which would improve some criterion without causing a simultaneous worsening in at least one other criterion. Other important definitions associated with Pareto optimality are the following:

Definition 3 (Pareto Dominance). A vector $\boldsymbol{u} = (u_1, \ldots, u_k)$ is said to dominate $\boldsymbol{v} = (v_1, \ldots, v_k)$ (denoted by $\boldsymbol{u} \preccurlyeq \boldsymbol{v}$) if and only if $\boldsymbol{u}$ is partially less than $\boldsymbol{v}$, i.e., $\forall i \in \{1, \ldots, k\}, \; u_i \leq v_i \; \wedge \; \exists i \in \{1, \ldots, k\} : \; u_i < v_i$.

Definition 4 (Pareto Optimal Set). For a given MOP $\boldsymbol{f}(\boldsymbol{x})$, the Pareto optimal set is defined as $\mathcal{P}^* = \{\boldsymbol{x} \in \Omega | \neg \exists \boldsymbol{x'} \in \Omega, \boldsymbol{f}(\boldsymbol{x'}) \preccurlyeq \boldsymbol{f}(\boldsymbol{x})\}$.

Definition 5 (Pareto Front). For a given MOP $\boldsymbol{f}(\boldsymbol{x})$ and its Pareto optimal set $\mathcal{P}^*$, the Pareto front is defined as $\mathcal{PF}^* = \{\boldsymbol{f}(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{P}^*\}$.

Obtaining the Pareto front of a MOP is the main goal of multiobjective optimization. However, given that a Pareto front can contain a large number of points, a good solution must contain a limited number of them, which should be as close as possible to the exact Pareto front, as well as they should be uniformly spread. Otherwise, they would not be very useful to the decision maker.

Algorithm 1 Pseudocode for a Canonical cGA

```
1: proc Steps_Up(cga)        //Algorithm parameters in 'cga'
2: while not Termination_Condition() do
3:    for individual ← 1 to cga.popSize do
4:       n_list←Get_Neighborhood(cga,position(individual));
5:       parents←Selection(n_list);
6:       offspring←Recombination(cga.Pc,parents);
7:       offspring←Mutation(cga.Pm,offspring);
8:       Evaluate_Fitness(offspring);
9:       Insert(position(individual),offspring,cga,aux_pop);
10:   end for
11:   cga.pop←aux_pop;
12: end while
13: end_proc Steps_Up;
```

3 The Algorithm

In this section we detail first a description of a canonical cGA; then, we describe the algorithm MOCell.

3.1 Cellular Genetic Algorithms

A canonical cGA follows the pseudo-code included in Algorithm 1. In this basic cGA, the population is usually structured in a regular grid of d dimensions ($d = 1, 2, 3$), and a neighborhood is defined on it. The algorithm iteratively considers as current each individual in the grid (line 3). An individual may only interact with individuals belonging to its neighborhood (line 4), so its parents are chosen among its neighbors (line 5) with a given criterion. Crossover and mutation operators are applied to the individuals in lines 6 and 7, with probabilities P_c and P_m, respectively. Afterwards, the algorithm computes the fitness value of the new offspring individual (or individuals) (line 8), and inserts it (or one of them) into the equivalent place of the current individual in the new (auxiliary) population (line 9) following a given replacement policy.

After applying this reproductive cycle to all the individuals in the population, the newly generated auxiliary population is assumed to be the new population for the next generation (line 11). This loop is repeated until a termination condition is met (line 2). The most usual termination conditions are to reach the optimal value, to perform a maximum number of fitness function evaluations, or a combination of both of them.

3.2 A Multiobjective cGA: MOCell

In this section we present MOCell, a multiobjective algorithm based on a cGA model. Its pseudo-code is given in Algorithm 1. We can observe that Algorithms 1 and 2 are very similar. One of the main differences between the two algorithms is the existence of a *Pareto front* (Definition 5) in the multiobjective case. The Pareto front is just an additional population (the external archive) composed of a number of the non-dominated solutions found, since it has a maximum size. In order to manage the insertion of solutions in the Pareto front with the goal of obtaining a diverse set, a density estimator based on the crowding distance (proposed for NSGA-II [10]) has been used. This measure is also used to remove solutions from the archive when this becomes full.

Algorithm 2 Pseudocode of MOCell

```
1: proc Steps_Up(mocell)       //Algorithm parameters in 'mocell'
2: Pareto_front = Create_Front() //Creates an empty Pareto front
3: while !TerminationCondition() do
4:    for individual ← 1 to mocell.popSize do
5:       n_list←Get_Neighborhood(mocell,position(individual));
6:       parents←Selection(n_list);
7:       offspring←Recombination(mocell.Pc,parents);
8:       offspring←Mutation(mocell.Pm,offspring);
9:       Evaluate_Fitness(offspring);
10:      Insert(position(individual),offspring,mocell,aux_pop);
11:      Insert_Pareto_Front(individual);
12:   end for
13:   mocell.pop←aux_pop;
14:   mocell.pop←Feedback(mocell,ParetoFront);
15: end while
16: end_proc Steps_Up;
```

MOCell starts by creating an empty Pareto front (line 2 in Algorithm 2). Individuals are arranged in a 2-dimensional toroidal grid, and the genetic operators are successively applied to them (lines 7 and 8) until the termination condition is met (line 3). Hence, for each individual, the algorithm consists of selecting two parents from its neighborhood, recombining them in order to obtain an offspring, mutating it, evaluating the resulting individual, and inserting it in both the auxiliary population (if it is not dominated by the current individual) and the Pareto front. Finally, after each generation, the old population is replaced by the auxiliary one, and a feedback procedure is invoked to replace a fixed number of randomly chosen individuals of the population by solutions from the archive.

We have incorporated a constrain handling mechanism in MOCell to deal with constrained problems. The mechanism is the same used by NSGA-II [10].

4 Computational Results

This section is devoted to the evaluation of MOCell. For that, we have chosen several test problems taken from the specialized literature, and, in order to assess how competitive MOCell is, we decided to compare it against two algorithms that are representative of the state-of-the-art, NSGA-II and SPEA2. Next, we briefly comment the main features of these algorithms, including the parameter settings used in the subsequent experiments.

The NSGA-II algorithm was proposed by Deb *et al.* [10]. It is characterized by a Pareto ranking of the individuals and the use of a crowding distance as density estimator. We have used Deb's NSGA-II implementation[1]. Specifically, we used the real-coded version of the algorithm and the parameter settings proposed in [10]. A crossover probability of $p_c = 0.9$ and a mutation probability $p_m = 1/n$ (where n is the number of decision variables) are used. The operators for crossover and mutation are SBX and polynomial mutation [9], with distribution indexes of $\eta_c = 20$ and $\eta_m = 20$, respectively. The population and archive sizes are 100 individuals. The algorithm stops after 25000 function evaluations.

In Table 1 we show the parameters used by MOCell. A square toroidal grid of 100 individuals has been chosen for structuring the population. The neighborhood used is composed

[1] NSGA-II is available for downloading at: `http://www.iitk.ac.in/kangal/soft.htm`

Table 1. Parameterization used in MOCell

Population Size	100 individuals (10×10)
Stopping Condition	25000 function evaluations
Neighborhood	1-hop neighbours (8 surrounding solutions)
Selection of Parents	binary tournament + binary tournament
Recombination	simulated binary, $p_c = 1.0$
Mutation	polynomial, $p_m = 1.0/L$ (L = individual length)
Replacement	rep_if_better_individual (NSGA-II crowding)
Archive Size	100 individuals
Density Estimator	crowding distance
Feedback	20 individuals

of nine individuals: the considered individuals plus those located at its North, East, West, South, NorthWest, SouthWest, NorthEast, and SouthEast (see Fig. 1c). We have also used SBX and polynomial mutation with the same distribution indexes as NSGA-II and SPEA2. Crossover and mutation rates are $p_c = 1.0$ and $p_m = 1/L$, respectively.

The resulting offspring replaces the individual at the current position if the latter is better than the former, but, as it is usual in multiobjective optimization, we need to define the concept of "best individual". Our approach is to replace the current individual if it is dominated by the offspring or both are non-dominated and the current individual has the worst crowding distance (as defined in NSGA-II) in a population composed of the neighborhood plus the offspring. For inserting the individuals in the Pareto front, the solutions in the archive are also ordered according to the crowding distance; therefore, when inserting a non-dominated solution, if the Pareto front is already full, the solution with a worst crowding distance value is removed. Finally, after each iteration, 20 randomly chosen individuals in the population are replaced by the 20 best solutions from the external archive according to the crowding distance (feedback mechanism).

4.1 Test Problems

We have selected for our tests both constrained and unconstrained problems that have been used in most studies in this area. Given that they are widely known, we do not include full details of them here for space constraints. They can be found in the cited references and also in books such as [7] and [8].

SPEA2 was proposed by Zitler *et al.* in [17]. In this algorithm, each individual has assigned a fitness value that is the sum of its strength raw fitness and a density estimation based on the distance to the k-th nearest neighbor. Like in the case of NSGA-II, we have used the authors' implementation of SPEA2[2]. The algorithm is implemented within the framework PISA [5]. However, the implementation of SPEA2 does not contain a constraint-handling management, so we were forced to modify the original implementation for including the same constraint mechanism used in NSGA-II and MOCell. We have used the following values for the parameters. Both the population and the archive have a size of 100 individuals, and the crossover and mutation operators are the same used in NSGA-II, using the same values concerning their application probabilities and distribution indexes. As in NSGA-II, the stopping condition is to compute 25000 function evaluations.

[2] SPEA2 is available at: `http://www.tik.ee.ethz.ch/pisa/selectors/spea2/spea2.html`

Table 2. Unconstrained test functions

Problem	Objective functions	Variable bounds	n
Schaffer	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	$-10^5 \leq x \leq 10^5$	1
Fonseca	$f_1(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{n} \left(x_i - \frac{1}{\sqrt{n}}\right)^2}$ $f_2(\boldsymbol{x}) = 1 - e^{-\sum_{i=1}^{n} \left(x_i + \frac{1}{\sqrt{n}}\right)^2}$	$-4 \leq x_i \leq 4$	3
Kursawe	$f_1(\boldsymbol{x}) = \sum_{i=1}^{n-1} -10e^{-0.2*\sqrt{x_i^2 + x_{i+1}^2}}$ $f_2(\boldsymbol{x}) = \sum_{i=1}^{n} (\lvert x_i \rvert^a + 5\sin x_i^b); a = 0.8; b = 3$	$-5 \leq x_i \leq 5$	3
ZDT1	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = g(\boldsymbol{x})[1 - \sqrt{x_1/g(\boldsymbol{x})}]$ $g(\boldsymbol{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT2	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = g(\boldsymbol{x})[1 - (x_1/g(\boldsymbol{x}))^2]$ $g(\boldsymbol{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT3	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = g(\boldsymbol{x})\left[1 - \sqrt{\frac{x_1}{g(\boldsymbol{x})}} - \frac{x_1}{g(\boldsymbol{x})}\sin(10\pi x_1)\right]$ $g(\boldsymbol{x}) = 1 + 9(\sum_{i=2}^{n} x_i)/(n-1)$	$0 \leq x_i \leq 1$	30
ZDT4	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = g(\boldsymbol{x})[1 - (x_1/g(\boldsymbol{x}))^2]$ $g(\boldsymbol{x}) = 1 + 10(n-1) + \sum_{i=2}^{n}[x_i^2 - 10\cos(4\pi x_i)]$	$0 \leq x_1 \leq 1$ $-5 \leq x_i \leq 5$ $i = 2, ..., n$	10
ZDT6	$f_1(\boldsymbol{x}) = 1 - e^{-4x_1}\sin^6(6\pi x_1)$ $f_2(\boldsymbol{x}) = g(\boldsymbol{x})[1 - (f_1(\boldsymbol{x})/g(\boldsymbol{x}))^2]$ $g(\boldsymbol{x}) = 1 + 9[(\sum_{i=2}^{n} x_i)/(n-1)]^{0.25}$	$0 \leq x_i \leq 1$	10

Table 3. Constrained test functions

Problem	Objective functions	Constraints	Variable bounds	n
Osyczka2	$f_1(\boldsymbol{x}) = -(25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2(x_4-4)^2 + (x_5-1)^2)$ $f_2(\boldsymbol{x}) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2$	$g_1(\boldsymbol{x}) = 0 \leq x_1 + x_2 - 2$ $g_2(\boldsymbol{x}) = 0 \leq 6 - x_1 - x_2$ $g_3(\boldsymbol{x}) = 0 \leq 2 - x_2 + x_1$ $g_4(\boldsymbol{x}) = 0 \leq 2 - x_1 + 3x_2$ $g_5(\boldsymbol{x}) = 0 \leq 4 - (x_3-3)^2 - x_4$ $g_6(\boldsymbol{x}) = 0 \leq (x_5-3)^3 + x_6 - 4$	$0 \leq x_1, x_2 \leq 10$ $1 \leq x_3, x_5 \leq 5$ $0 \leq x_4 \leq 6$ $0 \leq x_6 \leq 10$	6
Tanaka	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = x_2$	$g_1(\boldsymbol{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(x_1/x_2)) \leq 0$ $g_2(\boldsymbol{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5$	$-\pi \leq x_i \leq \pi$	2
ConstrEx	$f_1(\boldsymbol{x}) = x_1$ $f_2(\boldsymbol{x}) = (1 + x_2)/x_1$	$g_1(\boldsymbol{x}) = x_2 + 9x_1 \geq 6$ $g_2(\boldsymbol{x}) = -x_2 + 9x_1 \geq 1$	$0.1 \leq x_1 \leq 1.0$ $0 \leq x_2 \leq 5$	2
Srinivas	$f_1(\boldsymbol{x}) = (x_1-2)^2 + (x_2-1)^2 + 2$ $f_2(\boldsymbol{x}) = 9x_1 - (x_2-1)^2$	$g_1(\boldsymbol{x}) = x_1^2 + x_2^2 \leq 225$ $g_2(\boldsymbol{x}) = x_1 - 3x_2 \leq -10$	$-20 \leq x_i \leq 20$	2

The selected unconstrained problems include the studies of Schaffer, Fonseca, and Kursawe, as well as the problems ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6. Their formulation is provided in Table 2. The constrained problems are Osyczka2, Tanaka, Srinivas, and ConstrEx. They are described in Table 3.

4.2 Performance Metrics

For assessing the performance of the algorithms on the test problems, two different issues are normally taken into account: (i) minimize the distance of the Pareto front generated by the proposed algorithm to the exact Pareto front, and (ii) to maximize the spread of solutions found, so that we can have a distribution of vectors as smooth and uniform as possible. To determine the first issue it is usually necessary to know the exact location of the true Pareto

front; in this work we have obtained these fronts using an enumerative search strategy, and they are publicly available at `http://neo.lcc.uma.es/software/esam` (an exception are the ZDTx problem family, whose fronts can be easily computed because their solutions are known).

- **Generational Distance** This metric was introduced by Van Veldhuizen and Lamont [16] for measuring how far the elements are in the set of non-dominated vectors found so far from those in the Pareto optimal set, and it is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \ , \tag{1}$$

  where n is is the number of vectors in the set of non-dominated solutions, and d_i is the Euclidean distance (measured in objective space) between each of these solutions and the nearest member of the Pareto optimal set. It is clear that a value of $GD = 0$ means that all the generated elements are in the Pareto optimal set. In order to get reliable results, non-dominated sets are normalized before calculating this distance measure.
- **Spread** The *Spread* metric [10] is a diversity metric that measures the extent of spread achieved among the obtained solutions. This metric is defined as:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} \left| d_i - \bar{d} \right|}{d_f + d_l + (N-1)\bar{d}} \ , \tag{2}$$

  where d_i is the Euclidean distance between consecutive solutions, $\bar{d}$ is the mean of these distances, and d_f and d_l are the Euclidean distances to the *extreme* (bounding) solutions of the exact Pareto front in the objective space (see [10] for the details). This metric takes a zero value for an ideal distribution, pointing out a perfect spread out of the solutions in the Pareto front. We apply this metric after a normalization of the objective function values.

4.3 Discussion of the Results

The results are summarized in Tables 4 (GD) and 5 (Δ), and the best result for each problem has a grey colored background. For each problem, we carried out 100 independent runs, and the tables include the mean, $\bar{x}$, and the standard deviation, σ_n, of the results. Since we deal with stochastic algorithms, an statistical analysis of the results has been made. It consists of the following steps. First a Kolmogorov-Smirnov test is performed in order to check whether the values of the results follow a normal distribution or not. If so, an ANOVA I test is done, otherwise we perform a Kruskal-Wallis test. We always consider in this work a 95% confidence level in the statistical tests. Symbol '+' in tables 4 and 5 means that the differences among the values of the three algorithms for a given problem have statistical confidence (p-value under 0.05).

We consider first the metric GD (Table 4). We can observe that the three compared algorithms provide the best results for four out of the 12 studied problems. According to these results we cannot decide a winner algorithm considering convergence, although they allow us to conclude that MOCell is a competitive algorithm compared to NSGA-II and SPEA2. Indeed, if we consider only the constrained studied problems, the reader can see that MOCell behaves better than the other compared algorithms, since it reports the best

Table 4. Mean and standard deviation of the convergence metric GD

Problem	MOCell $\bar{x}_{\sigma_n}$	NSGA-II $\bar{x}_{\sigma_n}$	SPEA2 $\bar{x}_{\sigma_n}$	
Schaffer	2.408e-4$_{\pm 1.79e-5}$	2.328e-4$_{\pm 1.19e-5}$	2.365e-4$_{\pm 1.06e-5}$	+
Fonseca	1.983e-4$_{\pm 1.60e-5}$	4.683e-4$_{\pm 3.95e-5}$	2.251e-4$_{\pm 2.37e-5}$	+
Kursawe	1.435e-4$_{\pm 1.01e-5}$	2.073e-4$_{\pm 2.22e-5}$	1.623e-4$_{\pm 1.51e-5}$	+
Zdt1	4.057e-4$_{\pm 6.57e-5}$	2.168e-4$_{\pm 3.57e-5}$	1.992e-4$_{\pm 1.34e-5}$	+
Zdt2	2.432e-4$_{\pm 9.29e-5}$	1.714e-4$_{\pm 3.80e-5}$	1.095e-4$_{\pm 5.37e-5}$	+
Zdt3	2.540e-4$_{\pm 2.78e-5}$	2.199e-4$_{\pm 3.72e-5}$	2.336e-4$_{\pm 1.34e-5}$	+
Zdt4	8.273e-4$_{\pm 1.85e-3}$	4.888e-4$_{\pm 2.59e-4}$	6.203e-2$_{\pm 3.94e-2}$	+
Zdt6	2.106e-3$_{\pm 3.33e-4}$	1.001e-3$_{\pm 8.66e-5}$	8.252e-4$_{\pm 5.15e-5}$	+
ConstrEx	1.968e-4$_{\pm 2.29e-5}$	2.903e-4$_{\pm 3.19e-5}$	2.069e-4$_{\pm 1.78e-5}$	+
Srinivas	5.147e-5$_{\pm 1.51e-5}$	1.892e-4$_{\pm 3.01e-5}$	1.139e-4$_{\pm 1.98e-5}$	+
Osyczka2	2.678e-3$_{\pm 5.30e-3}$	1.071e-3$_{\pm 1.33e-4}$	6.149e-3$_{\pm 1.14e-2}$	+
Tanaka	7.494e-4$_{\pm 7.09e-5}$	1.214e-3$_{\pm 7.95e-5}$	7.163e-4$_{\pm 7.13e-5}$	+

results for ConstrEx and Srinivas, while it is the second best approach in the other two problems.

Regarding the spread metric (Table 5), the results indicate that MOCell clearly outperforms the other two algorithms concerning the diversity of the obtained Pareto fronts, since it yields the best values in 9 out of the 12 problems. Additionally, MOCell reports the best results for all the constrained studied problems. It stands out that NSGA-II can not obtain the best value for the spread metric in any problem.

In order to graphically show our results, we plot in Fig. 2 three fronts obtained by MOCell, NSGA-II, and SPEA2, together with the optimal Pareto set obtained by the enumerative algorithm, for problem ConstrEx. The selected fronts are those having the best diversity (lowest value of Δ) among the 100 produced ones by each technique for that problem. We can observe that the nondominated set of solutions generated by MOCell achieves an almost perfect spread out and convergence. Notice that SPEA2 obtains a very good diversity for values of $f_1(\boldsymbol{x})$ lower than 0.66 (similar to the solution of MOCell), but it finds only 10 solutions when $f_1(\boldsymbol{x}) \in [0.66, 1.0]$. Regarding NSGA-II, its front does not have that

Table 5. Mean and standard deviation of the diversity metric Δ

Problem	MOCell $\bar{x}_{\sigma_n}$	NSGA-II $\bar{x}_{\sigma_n}$	SPEA2 $\bar{x}_{\sigma_n}$	
Schaffer	2.473e-1$_{\pm 3.11e-2}$	4.448e-1$_{\pm 3.62e-2}$	1.469e-1$_{\pm 1.14e-2}$	+
Fonseca	9.695e-2$_{\pm 1.08e-2}$	3.596e-1$_{\pm 2.83e-2}$	1.445e-1$_{\pm 1.28e-2}$	+
Kursawe	4.121e-1$_{\pm 4.32e-3}$	5.460e-1$_{\pm 2.41e-2}$	4.390e-1$_{\pm 8.94e-3}$	+
Zdt1	1.152e-1$_{\pm 1.40e-2}$	3.645e-1$_{\pm 2.91e-2}$	1.684e-1$_{\pm 1.29e-2}$	+
Zdt2	1.120e-1$_{\pm 1.61e-2}$	3.644e-1$_{\pm 3.03e-2}$	1.403e-1$_{\pm 6.71e-2}$	+
Zdt3	6.998e-1$_{\pm 3.25e-2}$	7.416e-1$_{\pm 2.25e-2}$	7.040e-1$_{\pm 1.78e-2}$	+
Zdt4	1.581e-1$_{\pm 6.14e-2}$	3.651e-1$_{\pm 3.32e-2}$	1.049e-1$_{\pm 1.71e-1}$	+
Zdt6	1.859e-1$_{\pm 2.33e-2}$	2.988e-1$_{\pm 2.48e-2}$	1.728e-1$_{\pm 1.16e-2}$	+
ConstrEx	1.323e-1$_{\pm 1.32e-2}$	4.212e-1$_{\pm 3.52e-2}$	5.204e-1$_{\pm 1.58e-2}$	+
Srinivas	6.191e-2$_{\pm 8.63e-3}$	3.680e-1$_{\pm 3.02e-2}$	1.628e-1$_{\pm 1.25e-2}$	+
Osyczka2	2.237e-1$_{\pm 3.50e-2}$	4.603e-1$_{\pm 5.58e-2}$	3.145e-1$_{\pm 1.35e-1}$	+
Tanaka	6.629e-1$_{\pm 2.76e-2}$	7.154e-1$_{\pm 2.35e-2}$	6.655e-1$_{\pm 2.74e-2}$	+

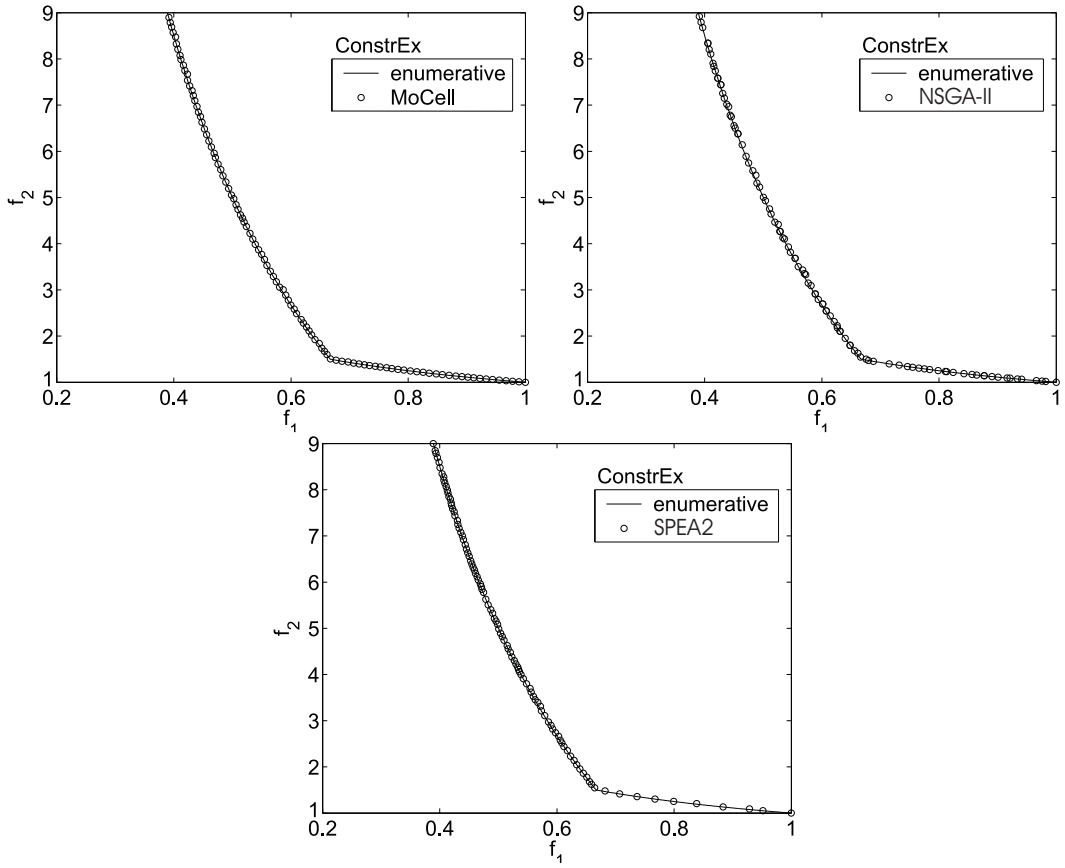


Fig. 2. MOCell finds a better convergence and spread of solutions than NSGA-II and SPEA2 on problem ConstrEx

problem, but its worst diversity with respect to the case of MOCell stands out at a simple glance.

We also want to remark that, concerning diversity, MOCell is not only the best of the three analyzed algorithms, but the differences in the spread values are in general noticeable compared to the rest of algorithms.

5 Conclusions and Future Work

We have proposed MOCell, a cellular genetic algorithm to solve multiobjective optimization problems. The algorithm uses an external archive to store the nondominated individuals found during the search. The most salient feature of MOCell with respect to the other cellular approaches for multiobjective optimization is the feedback of individuals from the archive to the population. MOCell was validated using a standard methodology which is currently used within the evolutionary multiobjective optimization community. The algorithm was compared against two state-of-the-art multiobjective optimization algorithms, NSGA-II and SPEA2; for that purpose, twelve test problems, including unconstrained and constrained

ones, were chosen and two metrics were used to assess the performance of the algorithms. The results of the metrics reveal that MOCell is competitive considering the convergence metric, and it clearly outperforms all the proposals on the considered test problems according to the spread metric.

Finally, the evaluation of MOCell with other benchmarks and its application to solve real-world problems are matter of future work.

Acknowledgments

This work has been funded by FEDER and the Spanish MCYT under contract TIN2005-08818-C04-01, the OPLINK project (`http://oplink.lcc.uma.es`).

References

1. E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, April 2005.
2. E. Alba, B. Dorronsoro, M. Giacobini, and M. Tomasini. *Handbook of Bioinspired Algorithms and Applications, Chapter 7*, chapter Decentralized Cellular Evolutionary Algorithms, pages 103–120. CRC Press, 2006.
3. E. Alba, B. Dorronsoro, F. Luna, A.J. Nebro, P. Bouvry, and L. Hogie. A Cellular Multi-Objective Genetic Algorithm for Optimal Broadcasting Strategy in Metropolitan MANETs. *Computer Communications*, To appear, 2006.
4. E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.
5. S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA - A Platform and Programming Language Independent Interface for Search Algorithms. In *EMO 2003*, pages 494–508, 2003.
6. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.
7. C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.
8. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2001.
9. K. Deb and R.B. Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995.
10. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
11. M. Kirley. MEA: A metapopulation evolutionary algorithm for multi-objective optimisation problems. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 949–956. IEEE Press, 2001.
12. J. Knowles and D. Corne. The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 9–105, Piscataway, NJ, 1999. IEEE Press.
13. M. Laumanns, G. Rudolph, and H.P. Schwefel. A Spatial Predator-Prey Approach to Multi-Objective Optimization: A Preliminary Study. In *Parallel Problem Solving from Nature V*, pages 241–249, 1998.
14. B. Manderick and P. Spiessens. Fine-grained parallel genetic algorithm. In *Proc. of the Third Int. Conf. on Genetic Algorithms (ICGA)*, pages 428–433, 1989.

15. T. Murata and M. Gen. Cellular Genetic Algorithm for Multi-Objective Optimization. In *Proc. of the 4th Asian Fuzzy System Symposium*, pages 538–542, 2002.
16. D.A. Van Veldhuizen and G.B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Dept. Elec. Comput. Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, 1998.
17. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), 2001.

Analysis and Comparison of Two Nature-Inspired Cooperative Strategies

Francisco Bonachela, David Pelta, and Alejandro Sancho Royo

Models of Decision and Optimization Research Group
Depto. de Ciencias de la Computación e I.A.
Calle Periodista Daniel Saucedo Aranda s/n
Universidad de Granada, 18071 Granada, Spain
dpelta@decsai.ugr.es

Abstract. *In recent years, biological and natural processes have been influencing the methodologies in science and technology in an increasing manner. In particular, the role that plays the cooperation among individuals is being studied more frequently and profoundly in diverse scopes of knowledge.*

We present here a system where a population of cooperative agents is applied to solve a particular problem using also a set of solutions. The focus is set on the cooperation mechanism: we took inspiration from Nature's examples of cooperation among unrelated individuals, namely reciprocity and byproduct mutualism.

We have simulated these strategies and we evaluated them over two aspects: the ability to solve a simple optimization problem and to maximize the lifetime of the agents.

1 Introduction

The influence of biology in computing can be traced back to the early days of Von Neumann, who use the brain as inspiration for modeling the first digital computer. Nowadays, at least three lines of research on biologically inspired computing can be defined: 1) the use of biology as a metaphor for algorithm's development, 2) the construction of information processing systems that use biological materials and 3) the effort to understand how biological organisms "compute". See for example [2,7,9] for more information on biologically inspired computing

Following the first line, we know that living beings can engage in different types of interaction, from altruistic cooperation to even open conflict. One specific kind of social interaction is cooperative problem solving (CPS), where a group of autonomous entities work together to achieve certain goal. The natural correspondence between autonomous entities and metaheuristics, and problem solving with an optimization problem, paves the way for the development of nature-inspired cooperative strategies.

In this contribution, we focus on the role that plays the cooperation between individuals not related by kinship. This type of cooperation is interesting because it cannot be understood directly by means of a strictly Darwinist sense. Studies in Behavioral Ecology, Ethology, Anthropology, Economy, Sociology... converge together with the classic tools of Game Theory and algorithmic and computer science technologies to analyze the consequences of cooperation in different contexts [5].

We describe here two examples of this kind of cooperation and we propose a model for their implementation and test. The basic idea is to use extremely simple agents that move around a grid of solutions for a particular problem. Agents vary the solutions and have a particular class determined by their type of cooperation.

The aim of this contribution is to compare and evaluate the role of different types of cooperation regarding problem solving. Using a simple optimization problem, we evaluate cooperative vs non-cooperative strategies, the effects that some parameters have over particular cooperative strategies, and finally we analyze what happens when agents from different classes are allowed to collaborate.

The organization of the paper to achieve the proposed objectives is as follows: in Section 2 different types of cooperation among unrelated individuals are described. Then in Section 3 the model and its implementation are explained. The computational experimentation has been divided in two parts. In Section 4 just one type of agent in the simulation is used and then in Section 5, the use of agents of different types simultaneously that may or may not cooperate is considered. Finally, Section 6 is devoted to discussions and final comments.

2 The Role of Cooperation

The role that plays the cooperation between individuals not related by kinship is being studied more frequently and profoundly in diverse scopes of the knowledge. Studies in Behavioral Ecology, Ethology, Anthropology, Economy, Sociology, etc., converge together with the classic tools of Game Theory and the algorithmic and computer science' technologies to analyze every time with more subtlety and depth the consequences of cooperation in different contexts. This type of cooperation is interesting because it cannot be understood directly in a strictly Darwinist sense [5].

In the context of cooperative strategies for optimization the use of a population of elements is a fundamental part. Such elements may be solutions to the problem at hand, "ants", "particles", "bees", "neurons", etc. and when using adequately, they form the basis of successful metaheuristics for problem solving [1, 3, 12].

On several models of population-based heuristics, a population of solutions is evolved by means of some process (Darwinist, dynamic, etc.) to obtain better and better solutions to the problem, like in evolutionary computation or parallel strategies [6, 8].

Another option, less investigated, is to have a population of operators or algorithms that traverse the search space (a set of solutions) cooperatively. When this cooperation is done through the environment, a scheme like the one of ant colony optimization is obtained (although in general, there is no population of solutions). Some works exists also in the field of multimemetic algorithms, where a set of local searchers are available during the search process [13] but collaboration among them is not considered. Some relation may be also found with cooperative search [10, 11] or multiple interacting walks [15].

When the cooperation among agents is done directly, a different scheme arises. In previous works we have explored the cooperation by means of centralized coordination and the role that plays the memory in this type of coordination [14].

Here, we want to explore decentralized cooperation mechanisms among individuals. From the specialized bibliography on cooperation in natural environments we find two interesting methods of cooperation between no relative individuals. We want to emphasize the interest in that the individuals are not relatives because in this type of cooperation no transmission of information from parents to children takes part, so the model does not put the emphasis in the reproductive processes.

We consider these schemes suitable to be applied when we have a set of agents operating simultaneously on the space of solutions and where the improvements made during the search

are not translated necessarily in mechanisms of representation of solutions that pass from parent to child.

The long term goal of this approach is that, through cooperation and self adjusting, the whole system adapts the population of solver agents to fit the requirements of the problem being solved.

As we stated before we will focus on two models of cooperation among unrelated entities: reciprocity and byproduct mutualism. For comparison purposes, we include a baseline model of "selfish" or independent behaviour. All the strategies are described below. Each agent has certain amount of energy or life available and we will refer to the "type" of an agent to indicate the cooperation strategy being used.

2.1 Independent agents

It is the situation when there is not cooperation at all. All the individuals of the species work alone and do not depend on what the other entities did, are doing or will do in the future. The only goal of one of this individuals is to try to improve the solution as much as possible by itself.

2.2 Cooperation through Reciprocity

A classical example of reciprocity in nature is Wilkinson's work [17] on blood sharing among vampire bats (Desmodus rotundus). In Dugatkin [4] we can read:

> "Vampire bats, typically live in groups composed largely of females, with a low average coefficient of relatedness [...] Somewhat remarkably, females in a nest of vampire bats, regurgitate blood meals to other that have failed to obtain food in the recent past"

This cooperation model has been implemented in our simulation in the following way: each agent of this breed or type, with a high amount of life will give some life to a closer and weaker agent (in terms of energy or life). The transferred amount of life is lost by the stronger agent, therefore the total amount of life remains the same after the deal.

2.3 Cooperation through byproduct Mutualism

This cooperation scheme is adapted from the following example quoted by Dugatkin [4] about the mutualism in house sparrows(Passer domesticus) that has been studied by Elgar in 1986 [6]

> "Elgar found evidence that those sparrows arriving at a patch of food first were the most likely to produce chirrup calls. Furthermore, chirrup call rates were higher when the food resource was divisible. When food items were small enough that sparrows could pick them and fly away, that is just what the sparrows did, and in the process, they produced no chirrup calls. Chirrup calls were, however, emitted when larger foods items -those that were too big to remove from the experimental area- were found at feeders."

This cooperation model has been implemented in our simulation in the following way: when an agent of this breed does a beneficial action (improves a solution in our case), performs a call to all other mutualist agents that are close in space. These agents go to the position where the caller agent is.

3 Model Definition and Implementation

We want to design a system where a set of agents solves a problem cooperatively, while their corresponding lifetime is maximized.

For demonstration purposes, we use a simple example of problem solving, namely real function optimization using the sphere model.

So, we are going to simulate and test the behavior of the previous cooperation models with two goals in mind:

1) Problem solving suitability We want to obtain a solution as good as possible for the sphere function over $\Re^n$:

$$\sum_{i=1}^{n} x_i^2 \tag{1}$$

Although this function is trivially solved with classical optimization techniques, it is enough for our demonstration purposes. The optimum value is 0 when $x_i = 0 \forall i \in 1..n$

2) Lifetime Maximization We want to keep our agents alive as much time as possible.

In order to model and implement our ideas we have used a freely distributed tool called NetLogo [16] that allows to check easily, quickly and interactively any working hypothesis about the simulation model.

The general way of operation of NetLogo is the following one: there is a central grid divided in squares or patches. These patches can store some information. There are also agents, that move around the grid, that can store information and modify the patches. With the suitable code we can simulate the cooperation models that we have explained above.

The simulation steps can be resumed in the following pseudo-code:

```
Initialization of variables
Evaluation of the patches
Do While (stop conditions not fulfilled)
[
    Movement of the agents (to a random neighbour patch)
    Evaluation of the variables of the patch
    Modification of the variables of the patch
    Comparison of the solutions
    Cooperation among agents
    Creation of new agents
    Update statistics and do plots
]
```

The interface of one of the simulations appears in Fig 1 where it can be seen the high amount of factors that can affect the simulation. It is quite a flexible model.

The center of the image shows a grid where each square or patch contains a list of values that represents a feasible solution for the problem at hand. The different levels of gray are

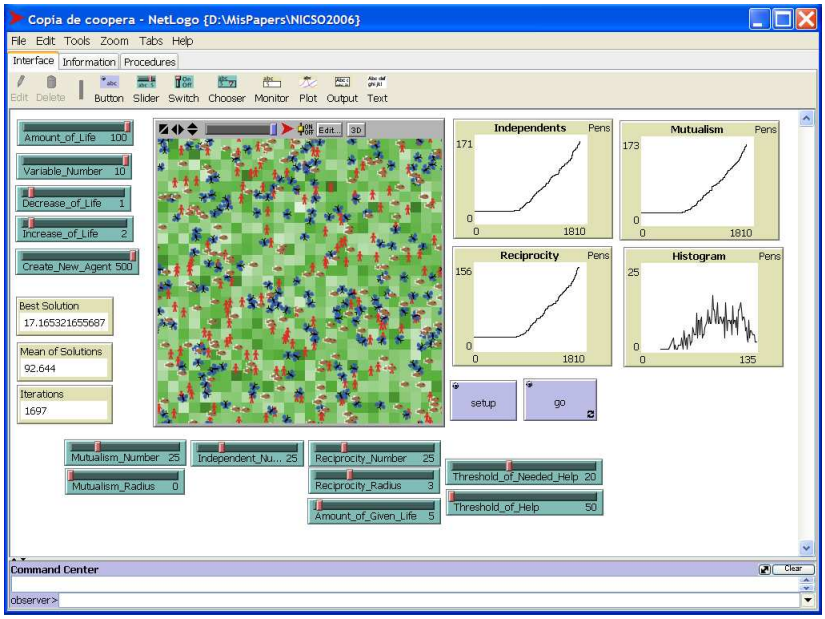


Fig. 1. Appearance of NetLogo in a simulation

proportional to the the fitness of the patch(solution). The clearer is the gray, the better is the solution.

When the simulation starts, patches (solutions) are randomly initialized.

The agents, representing individuals of different types or breeds, are born with a fixed *Amount-of-Life*. Each breed is represented with a different symbol.

Agents move around the grid, one patch every iteration. When an agent arrives to a patch, it takes the solution $X = \{x_1, x_2, \ldots, x_n\}$ and modifies the value of one randomly selected variable $x_i = x_i + \xi$ where ξ is a random number obtained from a normal distribution with parameters $mean = 0$ and $variance = 1$.

If the new value of the solution is better than the old one, the patch keeps the new solution and the agent earns life (according to the parameter *Increase-of-Life*). In the other case, patch's solution is kept and the agent loses life (according to the parameter *Decrease-of-Life*). Then, the colour of the patch is changed accordingly, therefore the grid tends to be brighter as the model evolves.

The agent's life gives raise to two additional behaviors:

- When an agent a_i achieves certain amount of life l higher than a parameter *Create-New-Agent*, then a_i will reproduce, giving raise to a new agent a_q of the same type. Now, the life of a_i is $l = l - Amount\text{-}of\text{-}Life$ and a_q starts with *Amount-of-Life* life.
- When an agent a_i loses all its life, it dies and dissapear from the grid

Besides the parameters described above, the model provides three additional parameters for setting up the initial number of agents of each type that will be available in the initial population. These parameters are: *Mutualist-Number*, *Reciprocity-Number* and *Independent-Number*.

When cooperative models are used, namely reciprocity and byproduct mutualism, the user can control the scope of the cooperation using the parameters *mutualism-radius* and *reciprocity-radius*. Any cooperative agent will search inside of the circle delimited by the radius to look for other agents to cooperate with.

The behavior of the system can be traced using a set of real time monitors, graphs and histograms. These new elements are:

- *Best Solution Monitor:* With this monitor we can know whenever we like what is the value of the best solution of the patches
- *Iterations Monitor:* It inform us about the number of iterations that have made the program
- *Mean of Solutions Monitor:* Mean of all the solutions present in the complete grid
- *Histogram Graph:*It shows us the evolution of the solutions of the grid. The shape of the graph tends to the left, where the solutions are next to zero.
- *Independents Graph:* It shows us the evolution of the number of independent agents.
- *Mutualism Graph:* It shows us the evolution of the number of mutualism agents.
- *Reciprocity Graph:* It shows us the evolution of the number of reciprocity agents.

3.1 Additional Implementation Details

The implementation of the cooperative strategies requires to take into account additional details that are described below for each strategy considered.

Details for cooperation through reciprocity The basic idea is that a strong agent will help a weaker one. In our model, the agents cooperate by giving life to the weakest agents. We can simulate this behaviour in the following way: reciprocity agents with more life than a limit (*Treshold-of-needed-help*) give part of his life (*Amount-of-Given-Life*) to agents with less life than a limit (*Treshold-of-Help*). All these three parameters add flexibility to the simulation since they can be modified by the user.

Sometimes happens that an agent has life enough to help other x agents, but there are y agents that need help (and $y > x$), then the weaker agents are helped first.

Details for cooperation through byproduct mutualism In this strategy, an agent that has improved a solution warns other ones that lies within certain radius. Those agents that "listened" the warning, jump from their current patch to the one where the caller agent is.

Instead of jumping, we also considered a step by step approach. However, on the way to the caller agent it may be warned from other position or it could even improve the current patch and call for other agents, thus forgetting the first call and so the mutualism behaviour would not be complete.

Jumping can cause also an undesirable situation: when using a high radius, an agent can call a lot of other agents and it would be easy to see a huge number of agents in the same patch. In the next iteration, one of the agents of this patch, can improve the solution and call all the agents that are in radius (at least all the agents in this patch would be called). If this behaviour is repeated, during all the iterations a lot of mutualist agents will stay in the same zone of the grid. To avoid this situation, after having been called, a mutualist agent can jump (at random direction) to escape from the patch from where it has been called.

4 Evaluation of Strategies

In this section we will evaluate the performance of the cooperative strategies proposed taking into account two factors:

1. the number of initial individuals ($iniPop$)
2. the radius of collaboration (δ)

We consider each possible combination of $(iniPop, \delta)$ with $iniPop \in \{20, 40, 60, 80\}$ and $\delta \in \{0, 1, 2, 3\}$. When $\delta = 0$ no collaboration/cooperation exists and the system essentially performs as a set of random walks on the patches. This case provides the baseline for comparison.

For each combination, we performed ten runs. The simulation finished when the whole set of individuals have disappeared. At the end of each run we recorded the best solution found and the number of iterations done (the so called "survival time"). The other initial values of parameters involved in the simulation are:

Amount-of-Life	$= 100$	*Decrease-of-Life*	$= 1$
Increase-of-Life	$= 2$	*Create-New-Agent*	$= 500$
Threshold-of-Needed-Help	$= 20$	*Amount-of-Life*	$= 100$
Amount-of-Given-Life	$= 5$	*Threshold-of-Help*	$= 40$

As it was previously stated, our goal is to obtain the best possible solution and the longest iterations (which means a higher rate of population survival).

4.1 Analysis of the "Byproduct Mutualism" Strategy

The results obtained using this strategy are shown in Fig 2.

Figure 2 (a) shows for each initial population size, how the average of the best solutions found vary in terms of the cooperation radius.

The analysis in this case is clear: as the radius of cooperation increases, the best solution found is improved. An Anova's test (T2 from Tamhane as implemented in SPSS© software) indicated that the differences on the average best solution among the different radius are statistically significant.

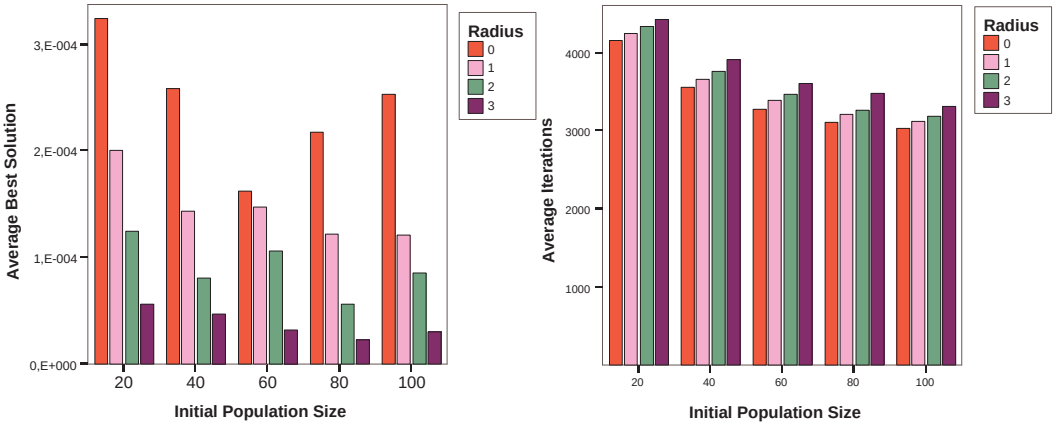
More important is the fact that the use of this cooperative strategy with any radius significantly improves the results obtained by an independent or selfish strategy (represented by $\delta = 0$).

If we focus on a particular radius, the influence of the initial population size is not clear and no conclusive statement can be made. Just when $\delta = 3$, we can observe a small improvement of the average best as the initial population grows.

Focusing on the second aspect to analyze, the iterations or survival time, we show the obtained results in Fig. 2 (b).

Again, we can detect a slightly increase on the average iterations value as the radius of collaboration grows. The differences are not very clear although the corresponding one between the average iterations is significant between $\delta = 3$ and $\delta = 0$ or 1.

This figure also reveals a very interesting behavior related with the size of the initial population. It can be observed that (considering a particular radius) the average iterations



(a) (b)

Fig. 2. Behavior of the system with a byproduct mutualism strategy in terms of the average best (a) and iterations (b)

diminishes as the initial population increases. In our simulations, the maximum number of agents that may exist simultaneously is fixed; under this situation, one would expect that having more individuals may ensure a longest survival. However, this is not the case.

We may understand these situation considering what happens with amount of energy available in the system. What the figure tells us is that it is better to start with a small population (low energy) to grow, adjust and gain energy slowly, than to start with a bigger population (higher initial amount of energy). The relation between the amount of energy and the number of agents in the system seems to be crucial for the success or failure of the simulation. More experimentation is under way to study this relation.

The previous analysis indicates that, when this strategy is used, the best radius is $\delta = 3$, independently of the initial population size chosen.

4.2 Analysis of the "Reciprocity" Strategy

The results obtained are shown in Fig. 3 and following the same scheme as in the previous experiment.

Observing Fig. 3 (a), the benefits of the use of a reciprocity strategy (as modeled here) are not clear. The differences among the use of different radius seems to diminish as the population size increase. It can be seen that the independent strategy (induced by $\delta = 0$) is not so bad when compared with $\delta = 1, 3$. However we can observe a slight improvement using $\delta = 2$ instead of $\delta = 0$ when the population starts with 20, 60 or 80 agents.

In terms of the average number of iterations performed, Fig. 3 (b) displays a similar behavior in relation to Initial Number of Individuals to the one shown in the previous strategy: as the initial population increases, the average iterations decrease. In this case, however, the average iterations also decrease when the radius of cooperation goes higher.

Again we may find an explanation in terms of how the energy of the system is managed in this cooperation strategy. Suppose we focus on an agent $\mathcal{A}$ having 5 units of energy to share with other weaker agents, and when the cooperation occurs, 1 unit is passed from $\mathcal{A}$ to the helped agent. When the radius of cooperation of cooperation is increased, the probability

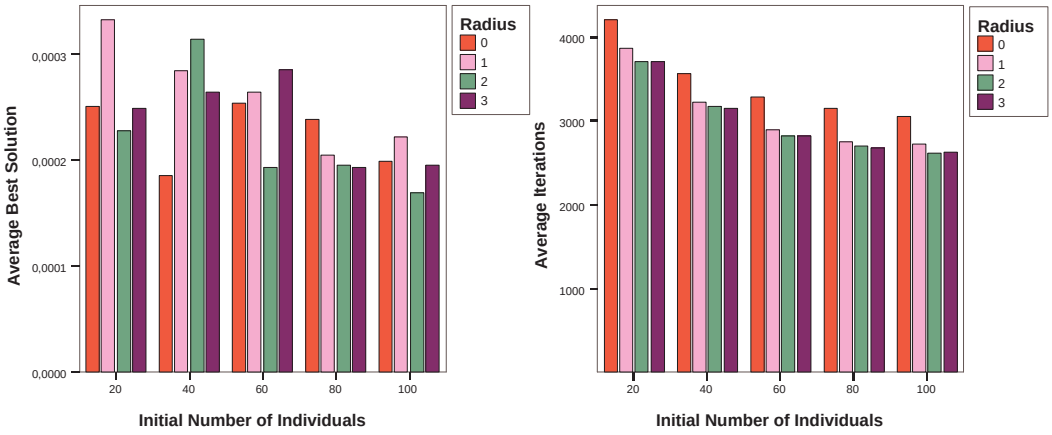


Fig. 3. Behavior of the system with a reciprocity strategy in terms of the average best (a) and iterations (b)

of finding weaker agents also increases. In the worst case, $\mathcal{A}$ may find 5 other agents to help, so each one will receive one additional unit of energy, while $\mathcal{A}$ will lose its corresponding 5 units. As it can be deduced from the simulations, it is worst to have 5 agents with one unit more of energy than one agent with 5 units less. Some sort of dissipation of energy occurs, that in turn, leads to a faster extinction of the whole population.

4.3 Byproduct Mutualism vs Reciprocity

To conclude the analysis of both strategies, we show in Fig 4 a comparison of the average best and iterations obtained by each strategy for every initial population size and radius of cooperation ($\delta = 0$ was omitted).

Considering both objectives, the comparison clearly indicates that a cooperation strategy inspired on byproduct mutualism is better than other one inspired in reciprocity. Moreover, a U-Mann-Whitney test indicates that differences on average best and average iterations are significant (over the whole set of results).

5 System's Behavior with mixed types of agents

In this second part we want to analyze what happens to the results when co-existence of different types of agents occurs. There may exist two or three types and in different proportions.

Two scenarios are designed for experimentation according to the fact that we can introduce cooperation among the different agent's types or, on the other hand, we can forbid any inter-class cooperation.

5.1 Inter-class cooperation disabled

In this case, agents of different types are placed on the grid, and they start to move following their own rules. Cooperation is done among agents of the same type.

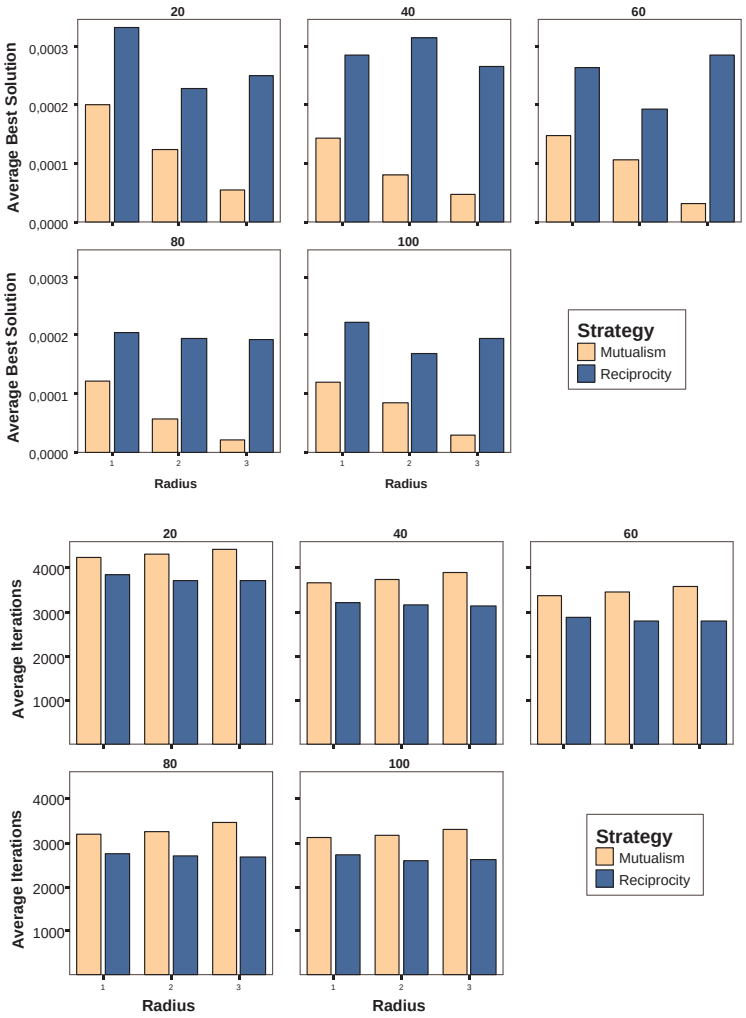


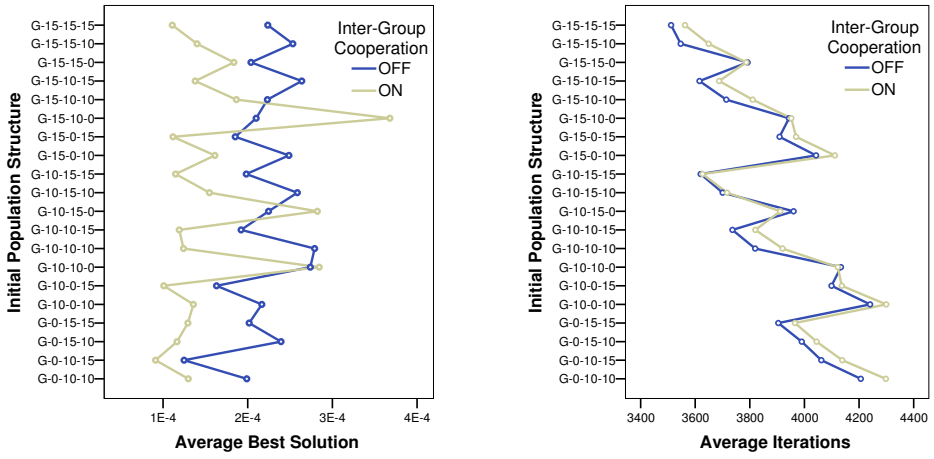
Fig. 4. Comparison of the Strategies by initial population size and radius of cooperation for average best (on top) and average iterations (bottom)

We should remark that an indirect cooperation occurs through the patches. When an agent $\mathcal{A}$ falls into a patch p with value γ, such value γ is the result of the previous actions performed by other agents on p, that in turn, may help or difficult $\mathcal{A}$' action.

5.2 Inter-class cooperation enabled

The first step in this situation is to define how collaboration is made among agents of different types. What we have done, is to setup a probability of cooperation between agents of different classes.

Agents following a byproduct mutualism strategy work as follows: when an agent improved a solution, it will call any other mutualist agent lying within the cooperation radius. Then every other non-mutualist agent may be called with a probability fixed in 0.2 .



(a) (b)

Fig. 5. Comparison of results when cooperation among different type of agents is allowed or not

Agents using reciprocity work in a similar manner. After helping the other reciprocity agents, and if it has enough energy available, it would help other agents of different type.

5.3 Computational Experiments

In order to analyze how the composition of the initial population and the role of cooperation among different agent's types affect the results, we design the following experiment.

Each initial population is defined as a triplet (n_I, n_R, n_M) where n_I is the number of non-cooperative or selfish agents, n_R is the number of agents using reciprocity and n_M is the number of agents using byproduct mutualism.

The values considered for each variable are $n_I, n_R, n_M \in \{0, 10, 15\}$. Triplets not including at least two types of agents are not considered. For each valid triplet we run ten repetitions of the simulation with and without inter-species cooperation. Each run finishes when the whole set of agents have disappeared.

Following the previous results, we set the radius for mutualism and reciprocity to 3 and 2 respectively.

Figure 5 (a) displays the results obtained. For every initial structure of the population, the average best value with and without inter-specie cooperation is shown. Groups follows a lexicographical order.

Immediately, one can observe that allowing cooperation among agents of different types produce a significant reduction of the average best. In other words, inter-specie cooperation allows to improve the results.

Just in 3 out of 20 cases, the cooperation produce worse results. These cases are $(15, 10, 0)$, $(10, 15, 0)$ and $(10, 10, 0)$: they do not have agents working with byproduct mutualism. A possible explanation for the deterioration of the average best is as follows: under these configurations, the stronger reciprocity agents need to share their energy with a higher number of other agents, which in turn leads to a kind of energy dissipation that does not produce a profitable result. In a sense, instead of reciprocity, we may understood the behav-

ior as "altruism" because a helper agent may give all its available energy to help others in very short time.

Groups having an amount of mutualist agents, perform rather differently. In terms of optimization, we may consider these agents as local improvers, because when an agent produces an improvement, it "calls" other agents that arrive to the same patch and alter the solution. Just by simple probabilities, the chance of such solution to be improved is increased.

Figure 5 (b) shows the results of the average number of iterations. In general, those groups having two types of agents, show a higher survival time than those with three independently of the cooperation factor.

Although the initial composition of the population seems to be determinant for increasing the survival time, the use of cooperation allowed for an additional increment.

6 Conclusions and Future Work

In this contribution we have presented a system where a population of cooperative agents is applied to solve a particular problem using also a set of solutions.

The focus is set on the cooperation mechanism: we took inspiration from Nature's examples of cooperation among unrelated individuals, namely reciprocity and byproduct mutualism. We have simulated these strategies and we evaluated them over two aspects: the ability to solve a simple optimization problem and to maximize the lifetime of the agents.

The experiments considering just one type of cooperation isolated allow us to conclude that when agents cooperate through byproduct mutualism, the results are significantly better (in terms of the best solution found and survival time) than those obtained when the cooperation is based on reciprocity.

Moreover, in the case of byproduct mutualism we found that increasing the cooperation radius we obtained improved results. This behaviour was not present when reciprocity is analyzed. In this case, and depending on the size of the initial population and the radius chosen, the results were worst than those of a non-cooperative strategy (where the agents did not cooperate at all).

We also considered initial populations where different set of agents (each one with its own way of cooperation) may coexist. The results are not conclusive with respect to populations with just one type of cooperation mechanism. However when agents using different strategies were allowed to cooperate, the results increased significantly.

Now, several lines of research should be explored. Firstly, we should develop a similar experimentation with a set of hard optimization functions to checkout if the current conclusions can be generalized. Secondly, the dynamics of the simulation may give us information to understand "why" a strategy is better than the other. The results indicated that it is better to start with a small population, so in particular, the distribution of energy or life along the simulation is crucial. Finally, we may consider time varying cost functions to analyze if a system composed by several agents using several cooperation strategies (intra and inter groups) may self-adapt to the "environmental" changes.

Acknowledgments

This work was supported in part by project TIN2005-08404-C04-01 from the Spanish Ministry of Science and Education and TIC-00129-PE.

References

1. A. Abraham, C. Grosan, and V. Ramos. *Stigmergic Optimization*, volume 31 of *Studies in Computational Intelligence*. Springer-Verlag, 2006.
2. L. N. de Castro and F. J. V. Zuben. *Recent Developments in Biologically Inspired Computing*. Idea Group Publishing, 2004.
3. M. Dorigo and T. Stützle. *Ant Colony Optimization*. The MIT Press, 2004.
4. L. Dugatkin. Animal cooperation among unrelated individuals. *Naturwissenschaften*, 89:533–541, 2002.
5. L. Dugatkin and H. Reeves, editors. *Game Theory and Animal Behavior*. Oxford University Press, 2000.
6. E. A. (Ed.). *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.
7. A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley and Sons, 2006.
8. D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press Series on Computational Intelligence. Wiley-IEEE Press, 2005.
9. N. Forbes. *Imitation of Life: How Biology is Inspiring Computing*. MIT Press, 2005.
10. T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *Proc. of AAAI93*, pages 231–236, Menlo Park, CA, 1993. AAAI Press.
11. B. Huberman. The performance of cooperative processes. *Physica D*, 42:38–47, 1990.
12. R. Martí, editor. *Handbook of Metaheuristics*. Kluwer Academic, 2003.
13. N.Krasnogor and S. Gustafson. A study on the use of "self-generation" in memetic algorithms. *Natural Computing*, 3(1):53–76, 2004.
14. D. Pelta, A. Sancho-Royo, C. Cruz, and J. L. Verdegay. Using memory and fuzzy rules in a cooperative multithread strategy for optimization. *Information Sciences*, 176(13):1849–1868, 2006.
15. M. Verhoeven and E. Aarts. Parallel local search. *Journal of Heuristics*, 1(1):43–65, 1995.
16. U. Wilensky. *NetLogo*. Center for Connected Learning and Computer- Based Modeling., Northwestern University, Evanston, IL, 1999. http://ccl.northwestern.edu/netlogo.
17. G. Wilkinson. Reciprocal food sharing in vampire bats. *Nature*, 208:181–184, 1984.

A New Ant Algorithm for Graph Coloring

***Alain Hertz*[1] *and Nicolas Zufferey*[2]**

[1] Département de mathématiques et de génie industriel, École Polytechnique de Montréal, Canada, alain.hertz@gerad.ca

[2] Corresponding author, Département d'Opérations et Systèmes de Décision, Faculté des Sciences de l'Administration, Université Laval, Québec (QC), G1K 7P4, Canada, nicolas.zufferey@fsa.ulaval.ca

Abstract. *Let $G = (V, E)$ be a graph with vertex set V and edge set E. The k-coloring problem is to assign a color (a number chosen in $\{1, \ldots, k\}$) to each vertex of V so that no edge has both endpoints with the same color. We describe in this paper a new ant algorithm for the k-coloring problem. Computational experiments give evidence that our algorithm is competitive with the existing ant algorithms for this problem, while giving a minor role to each ant. Our algorithm is however not competitive with the best known coloring algorithms*

1 Introduction to graph coloring

The graph coloring problem (GCP for short) can be described as follows. Given a graph $G = (V, E)$ with vertex set V and edge set E, and given an integer k, a *k-coloring* of G is a function $col : V \longrightarrow \{1, \ldots, k\}$. The value $col(x)$ of a vertex x is called the *color* of x. Vertices with a same color define a *color class*. If two adjacent vertices x and y have the same color, then vertices x and y are called *conflicting vertices* and the edge linking x with y is called a *conflicting edge*. A color class without conflicting edge is called a *stable set*. A k-coloring without conflicting edges is said to be *legal* and corresponds to a partition of the vertices into k stable sets. The GCP is to determine the smallest integer k (called the *chromatic number* of G) such that there exists a legal k-coloring of G. The GCP is NP-hard [15]. Exact solution methods can solve problems of relatively small size (no more than 100 vertices) [2], [3], [23], [18]. Upper bounds on the chromatic number can be obtained for larger instances by using heuristic algorithms.

The GCP has many practical applications such as the creation of timetables, frequency assignment, scheduling, design and operation of flexible manufacturing systems (e.g. [22], [14], [25]). As a result of its importance and simplicity, it is one of the most studied NP-hard problem, thus many methods have been developed to solve or to approach the GCP.

Given a fixed integer k, we consider the optimization problem, called k-GCP, which aims to determine a k-coloring of G that minimizes the number of conflicting edges. If the optimal value of the k-GCP is zero, this means that G has a legal k-coloring. The chromatic number of G can be determined by first computing an upper bound on this number (for example by means of a constructive method) and then by solving a series of k-GCPs with decreasing values of k until no legal k-coloring can be obtained. Many local search methods have been proposed to solve the k-GCP. For example, a tabu search is described in [19], and simulated annealing algorithms can be found in [5] and in [20]. Hybrid evolutionary

heuristics have also been successfully applied to this problem (e.g., [7], [13], [16]). For a recent survey, the reader is referred to [17].

In this paper, we propose a new kind of ant algorithm for the GCP. Our main goal is to show that even if we give a minor role to each ant, we can still obtain competitive results in comparison with other ant heuristics for the GCP.

2 Ant algorithms and two existing adaptations to the GCP

Evolutionary heuristics encompass various algorithms such as genetic algorithms, scatter search, ant systems and adaptive memory algorithms [4]. They can be defined as iterative procedures that use a central memory where information is collected during the search process.

Ant colonies were introduced in [11] and in [8], and are derived from the observation of ants in the nature: ants are able to find a shortest path between a food source and the nest using a trail system (called pheromone). In these methods, the central memory is modeled by a trail system. In the usual *ant system*, a population of ants is used, where each ant is a constructive heuristic able to build a solution step by step. At each step, an ant adds an element to the current partial solution. Each decision or *move* m is based on two ingredients: the greedy force (short term profit for the considered ant) $GF(m)$ and the trails $Tr(m)$ (information obtained from other ants). Let M be the set of all the possible moves. The probability $p_k(m)$ that ant k chooses move m is given by

$$p_k(m) = \frac{GF(m)^\alpha \cdot Tr(m)^\beta}{\sum_{m' \in M_k(adm)} GF(m')^\alpha \cdot Tr(m')^\beta},$$

where α and β are parameters and $M_k(adm)$ is the set of admissible moves that ant k can perform. When each ant of the population has build a solution, the trails can be updated for example as follows:

$$Tr(m) = \rho \cdot Tr(m) + \Delta Tr(m), \ \forall m \in M,$$

where $0 < \rho < 1$ is a parameter representing the evaporation of the trails, which is generally close or equal to 0.9, and $\Delta Tr(m)$ is a term which reinforces the trails left on move m by the ant population. That quantity is usually proportional to the number of times the ants performed move m, and to the quality of the obtained solutions when move m has been performed. More precisely, $\Delta Tr(m) = \sum_k \Delta Tr_k(m)$, where $\Delta Tr_k(m)$ is proportional to the quality of the obtained solutions when performing m. In some systems, the trails are updated more often (e.g. each time a single ant has built its solution [10]). In *hybrid ant systems*, the solutions provided by some ants may be improved using a local search technique. In the *max-min ant systems* [26], the authors proposed to normalize $GF(m)$ and $Tr(m)$ in order to better control these ingredients and thus the search process. An overview of ant algorithms can be found in [9].

Two ant heuristics for the GCP already exist in the literature. The first one was proposed in [6]. In their method, each ant is a constructive heuristic derived from *Dsatur* [2] or from

RLF [22]. Let us denote these methods by *ANT-DSATUR* and *ANT-RLF*, respectively. A move always consists in selecting a vertex and giving a color to it. The trail system is modeled using a matrix $Tr(x, y)$ proportional to:

- the number of times vertices x and y have the same color in the solutions provided by the ants;
- the quality of the solutions where $col(x) = col(y)$.

Another ant coloring method was recently proposed in [24], where each ant is a local search instead of a constructive heuristic. The authors consider (not necessarily legal) k-colorings and try to minimize the number of conflicts. The trail system is modeled, at iteration t, by the use of an auxiliary graph $G' = (V, E'(t))$ obtained from the original graph $G = (V, E)$ by adding edges. The edge set $E'(t)$ such that $E \subseteq E'(t)$ is updated at each iteration t in the following way. If many ants give a different color to x and y such that edge $[x, y] \notin E$, then $[x, y]$ is added to E'. A move consists in changing the color of a conflicting vertex in G, while minimizing the number of conflicts associated with the auxiliary graph G'.

3 General approach and role of the ants

The new proposed ant coloring method differs from those in [6] and [24] in that the role of each single ant is not to build a whole solution but only to contribute to give a color to a single vertex. Also, on the contrary to the existing ant coloring algorithms, we only make one solution evolve using a population of ants. We deal with (not necessarily legal) k-colorings and try to minimize the number f of conflicts. We always associate:

- a color in $\{1, \ldots, k\}$ to each ant;
- k ants to each vertex.

Thus, we use $|V|$ ants of each color. From a distribution of the ants on the vertices, we deduce a coloring of the graph using a procedure, called *Color*, which is derived from *Dsatur* [2] which we now describe.

Let A be a set of vertices which is initialized to V. At each step, *Dsatur* colors one vertex in A as follows:

1. select the vertex $x \in A$ with the largest *saturation degree*, which is the number of different colors that are adjacent to x; if more than one vertex maximize the saturation degree, the vertex with the largest number of adjacent vertices is chosen (ties are broken randomly);
2. assign the smallest color to x without creating any conflict, and remove x from A.

Procedure *Color* differs from *Dsatur* as follows. Given a set A of vertices to be colored, and given a k-coloring of the induced subgraph $G' = (V' = V - A, E')$ of G, the selection of the next vertex x in A to be colored is done as in *Dsatur*, but we assign to x a color that is represented by at least one ant on x, and among these colors, we choose the one that minimizes the number of conflicts. If several colors are possible, we give to x the color which is the most represented by the ants on x (ties are broken randomly).

At the beginning of our method, we place one ant of each color on every vertex and we set $A = V$. Thus, the first iteration of our method corresponds to an application of $Dsatur$. Then, we iteratively modify the positions of the ants on the graph and, at the end of each iteration, we recolor a subset $A \subseteq V$ of vertices, using the *Color* procedure.

Note that one might propose, in the *Color* procedure, to always select the next vertex to color as the vertex x in A such that x has the smallest number of colors represented by the ants on it. Suppose for example that only ants of color 2 are on x. In this case, it seems appropriate to color x as soon as possible because we can only give color 2 to it. This strategy was tested but its performance was very poor. This is due to the fact that such a strategy is not able to diversify the search process. Indeed, a vertex with very few colors represented on it will always be colored first (or among the first ones), and the *Color* procedure will probably always assign the same color to it.

Let a_i and a_j be two ants with colors i and j, respectively. Suppose that a_i is on vertex x and a_j is on vertex y. A *move* $m = (x, i) \leftrightarrow (y, j)$ consists in exchanging the positions of a_i and a_j on vertices x and y. Thus, a_i is moved from x to y and a_j is moved from y to x. At each iteration t of the algorithm, we modify the distribution of the ants on the graph by performing a sequence of such moves, and we reassign a color to some vertices by applying the *Color* procedure. We can already notice that each ant taken individually has an influence only on the vertex on which it is laying. This is not a significant role at all! At the end of each iteration t, it is important to reassign a color to:

1. all vertices x with a different set of ants on it (because it is forbidden to give a color to x which is not represented by at least one ant on x);
2. all conflicting vertices (because the goal of one iteration is to try to remove some conflicts).

Thus, at iteration t, we use the following set A in the *Color* procedure (recall that A is initialized to V):

$$A = \{ \text{ vertices involved in a move at iteration } t \} \cup \{ \text{ conflicting vertices at iteration } t-1 \}.$$

An important issue is to define the set of moves to perform at iteration t (for any fixed t). Let $N_l(x, t-1)$ be the number of ants of color l on x at iteration $t-1$. The goal of an iteration t is to change the color of at least one randomly chosen conflicting vertex x. Suppose x has color i at iteration $t-1$, thus $N_i(x, t-1) > 0$ (otherwise the *Color* procedure would not be able to assign color i to x at the end of iteration $t-1$). In order to be sure that we change the color of x using *Color* at the end of iteration t, we remove all ants of color i from x, i.e. we perform a sequence of $N_i(x, t-1)$ moves m, chosen in the set

$$M(t) = \{m = (x, i) \leftrightarrow (y, j) \text{ such that } y \neq x,\ N_j(y, t-1) > 0,\ j \neq i\}.$$

Note that in order to reduce the risk of cycling, if we remove all the ants of color i from vertex x, then we forbid to put an ant of color i on x during tab iterations, where $tab = \text{UNIFORM}(0, 9) + 0.6 \cdot NCV(s)$, where s is the current solution and $NCV(s)$ is the number of conflicting vertices in s. The choice of such tab value is the one proposed in the efficient tabu search coloring algorithm described in [16], which is derived from the tabu search proposed

in [19].

Another important issue is to define the way to select a move. As in most ant algorithms, we choose at iteration t a move m according to the greedy force $GF(m,t)$ and the trail $Tr(m,t)$. Given α and β, we propose to always choose the move m that maximizes

$$p(m,t) = \alpha \cdot GF(m,t) + \beta \cdot Tr(m,t),$$

where $GF(m,t)$ and $Tr(m,t)$ are normalized in $[0;1]$. We use normalized quantities in order to better control the weights of these ingredients during the process, as in some *max-min ant systems* [26].

3.1 Definition of the greedy force

Recall that the greedy force represents the short-term profit of a single ant. At each iteration, the short term profit consists in removing some conflicts. In order to remove a conflict, we should remove some ant-conflicts, where an *ant-conflict* occurs when two ants of the same color c are placed on two adjacent vertices x and y. Remember that in this case, the *Color* procedure may give the same color c to x and y, because color c is represented on x and y. Thus, a move with a large greedy force value should have the potential to significantly reduce the number of ant-conflicts, and consequently the number of conflicts. Suppose we aim to change the current color i of vertex x. We thus have to remove all the ants of color i from vertex x. In order to do that, we perform a sequence of moves of type $m = (x,i) \leftrightarrow (y,j)$. For such a move m, we define the greedy force $GF(m,t)$ by setting

$$GF(m,t) = Adv(m,t) - Disadv(m,t),$$

where $Adv(m,t)$ and $Disadv(m,t)$ are respectively the advantage and disadvantage of performing move m (i.e. exchanging ants a_i and a_j) at iteration t. Note that we always normalize Adv and $Disadv$ in interval $[0;1]$.

Let $S(x,y,i,t)$ be the number of ants of color i on vertices, different from y, which are adjacent to x. An ant a_i placed on vertex x is attracted by vertex y if:

1. there are several ants of color i on y;
2. there are several ants of color i on vertices, different from y, which are adjacent to x.

Similar considerations hold for the ant a_j placed on vertex y which is candidate to be placed on vertex x instead of a_i. Thus, we can set

$$\begin{aligned} Adv(m,t) = {} & N_i^2(y,t-1) + S(x,y,i,t-1) \\ & + N_j^2(x,t-1) + S(y,x,j,t-1). \end{aligned}$$

Note that we respectively use $N_i^2(y,t-1)$ and $N_j^2(x,t-1)$ instead of $N_i(y,t-1)$ and $N_j(x,t-1)$ in order to give more importance to the fact that ants with the same color should be grouped together. In addition, preliminary experiments showed us that such a formula leads to better results.

An ant a_j placed on vertex y is not attracted by vertex x if:

1. there are several ants of color j on y;
2. there are several ants of color j on vertices, different from y, which are adjacent to x.

As we know that all the ants of color i will be removed from x, vertex x is not attracted by vertex y only if there are several ants of color i on vertices, different from x, which are adjacent to y. Thus, we can set

$$Disadv(m,t) = N_j^2(y,t-1) + S(x,y,j,t-1) + S(y,x,i,t-1).$$

3.2 Definition of the trail system

We first define the way to update the trail left by ants of color c on vertex v at the end of iteration t:

$$tr(v,c,t) = \rho \cdot tr(v,c,t-1) + \Delta tr(v,c,t),$$

where $0 < \rho < 1$ is an evaporation parameter and the reinforcement value of color c on vertex v is $\Delta tr(v,c,t)$. Note that evaporation and reinforcement may simultaneously occur, as in any ant system.

We consider several cases. Suppose that during iteration t, the following events occurred:

1. x loses $p = N_i(x,t-1)$ ants of color i;
2. we put p ants on x, and the p ants have colors in the subset $C(x)$ of $\{1,\ldots,k\}$;
3. the p new ants on x came from a subset of vertices $P(x)$ of V.

A good trail system has to incorporate the following information:

1. if an ant of color c leaves vertex v, then $tr(v,c,t)$ should be evaporated (this is especially true for $v = x$ and $c = i$) and not reinforced;
2. if no ant of color c leaves a vertex $v \in P(x)$, then $tr(v,c,t)$ should be evaporated and slightly reinforced;
3. if an ant of color c arrives on vertex v, then $tr(v,c,t)$ should be evaporated and reinforced, and the reinforcement should be especially large if the generated solution s' from s is better than s, or better than s^*, which is the best solution visited so far during the search process;
4. if nothing occurs relatively to a vertex v and a color c, then $tr(v,c,t)$ should be slightly evaporated and not reinforced.

Preliminary experiments showed that the following parameter setting is appropriate.

- $\rho = 0.9$ and $\Delta tr(v,c,t) = 2 \cdot \delta$ if $v = x, c \in C(x)$;
- $\rho = 0.8$ and $\Delta tr(v,c,t) = 0$ if $v = x, c = 1$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = 0$ if $v = x, c \notin C(x) \cup \{i\}$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = 0$ if $v \in P(x), c$ such that there is an ant a_c which left v;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = \delta$ if $v \in P(x)$ and $c = i$;
- $\rho = 0.9$ and $\Delta tr(v,c,t) = \frac{\delta}{2}$ if $v \in P(x)$ and c such that there is no ant a_c which left v;
- $\rho = 0.99$ and $\Delta tr(v,c,t) = 0$ if $v \notin \{x\} \cup P(x)$ and $c \in \{1,\ldots,k\}$;

$$\text{where } \delta = \begin{cases} 0.1 \text{ if } f(s) - f(s') \geq 0 \\ 0.2 \text{ if } f(s) - f(s') < 0 \\ 0.4 \text{ if } f(s) - f(s^*) < 0 \end{cases}$$

Remember that $f(s)$ is the number of conflicts in solution s. We can now define the trail of a move m at iteration t as follows:

$$Tr[m = (x,i) \leftrightarrow (y,j), t] = tr(x,j,t) + tr(y,i,t) - tr(x,i,t) - tr(y,j,t).$$

Note that we initialize $tr(x,j,0) = 1,\ \forall x \in \{1,\ldots,n\},\ \forall j \in \{1,\ldots,k\}$, and we always normalize the tr values in interval $[0;1]$.

4 General algorithm

We have now all the ingredients necessary to formulate a new ant heuristic for the k-GCP. The main loop of our method, called $ANTCOL$, stops when a maximum number $MaxIter$ of iterations without improvement of the best solution s^* encountered so far have been performed. The algorithm is the following:

1. set $t = 0$;
2. place one ant of each color on each vertex;
3. initialize the trails of each color on each vertex to 1;
4. apply the $Color$ procedure with $A = V$; let s be the so obtained solution;
5. set $t = 1$, $t' = 1$, $A = V$, $f^* = f(s)$, and $s^* = s$;
6. **while** $t' < MaxIter$ **and** $f^* > 0$**, do**
   (a) determine the set $M(t)$ of the possible non tabu moves for iteration t;
   (b) compute the greedy force $GF(m,t)$ and the trail $Tr(m,t)$, $\forall \in M(t)$;
   (c) normalize the greedy forces and the trails in $[0,1]$;
   (d) perform the non tabu move $m \in M(t)$ maximizing $p(M,t)$; let $m = (x,i) \leftrightarrow (y,j)$ be such a move;
   (e) while there is at least one ant of color i on x, perform the move m maximizing $p(M,t)$ among the moves in $\{m = (x,i) \leftrightarrow (y,j) \in M(t) \mid y \neq x$ and color j is represented on $y\}$;
   (f) update the trails $tr(v,c)$ for each vertex v and each color c, then normalize those quantities;
   (g) update the tabu status;
   (h) set $A =$ { vertices involved in a move performed at iteration t }$\cup$ { conflicting vertices at iteration $t-1$ };
   (i) update the colors of the vertices in A using the $Color$ procedure; let s be the so obtained solution;
   (j) if $f(s) < f^*$, set $s^* = s$, $f^* = f(s)$, and $t' = -1$;
   (k) set $t = t + 1$ and $t' = t' + 1$;

Note that preliminary experiments showed that the use of $MaxIter = 5000$ is appropriate: the method is usually not able to improve the solution s^* further if we use a larger value. Also, preliminary experiments showed that $\alpha = 1$ and $\beta = 5$ are appropriate values in the following formula: $p(m,t) = \alpha \cdot GF(m,t) + \beta \cdot Tr(m,t)$. Thus, as we use normalized values for the trails and the greedy forces, it is better to give more importance to the trails. The reader interested in having more details on the parameter settings is referred to [27].

5 Obtained results and conclusion

We will compare our ant coloring algorithm with some other coloring heuristics. We choose to consider the following methods:

- *Dsatur* [2] which is a constructive algorithm,
- the ant algorithms *ANT-DSATUR* and *ANT-RLF* [6] (see Section 2);
- *Tabucol*, which is a very well-known tabu search algorithm originally proposed in [19] and improved in [16];
- *GH*, which is an hybrid genetic algorithm proposed in [16], and is now widely considered as the best coloring heuristic; such a method uses *Tabucol* as intensification procedure.

Note that the ant coloring heuristic proposed in [24] will not be considered because of the following reasons. First, the authors mainly tested their method on very easy instances, namely *le*450_5*a*, *le*450_5*b*, *le*450_5*c*, *le*450_5*d*, *le*450_15*a* and *le*450_15*b*, which are well-known benchmark instances [21]. But it is known that these instances can be optimally colored by exact algorithms within a few seconds! Then, they tested their algorithm on non standard random graphs they generated with a very small number of edges. However, no results is given for standard random graphs, and we have not been able to re-implement their algorithm. Finally, notice that in their method, each ant is a whole local search coloring heuristic, which means that, on the contrary to our method, an ant has a very significant role. Consequently, it is not relevant to compare their hybrid algorithm with our method. However, we compare our heuristic with another type of local search, namely *Tabucol*, which is considered as one of the most simple and efficient local search coloring algorithm [17].

The *density* of a graph is the average number of edges between two vertices. We compare the above methods on several random graphs of size $|V| \in \{100, 300, 500, 1000\}$ and density $d = 0.5$. These graphs are obtained by linking a pair of vertices by an edge with probability 0.5, independently for each pair. It is known in the graph coloring community that random graphs with $d = 0.5$ are hard to color. While the results obtained on these graphs are representative, the reader interested in having more details on the results obtained by *ANTCOL* and many other coloring heuristics on other instances is referred to [27].

All the experiments were done on a computer *Silicon Graphics Indigo2 (195 MHz, IP28 processor)*. We will not present the detailed CPU times for each graph because of their large variability. However, in order to give an idea of these CPU times, we mention that the time needed by *ANTCOL* to find a legal coloring varies from a few seconds to a few minutes if $|V| = 100$, from a few minutes to an hour if $|V| = 300$, from one to three hours if $|V| = 500$, and from three to eight hours if $|V| = 1000$. These computing times are comparable with the ones needed by the other methods. Once again, the reader interested in getting more detailed is referred to [27].

The results associated with *ANT-DSATUR* and *ANT-RLF* are taken from [6]. For the other methods, we generated four graphs for each considered value of $|V|$ and performed four runs on each graph. The results are summarized in Table 1. For each size $|V|$, we give the "most likely" number of colors used by each method to generate legal colorings. By "most likely", we mean the number of colors for which the success rate was at least 50%. In brackets, we mention the smallest number of colors found by each method on at least one

graph. For example, for $|V| = 300$, $ANTCOL$ will most likely build legal 39-colorings, and will sometimes generate legal colorings using only 37 colors.

$\|V\|$	ANT-RLF	ANT-DSATUR	DSATUR	Tabucol	GH	ANTCOL
100	16 (15)	16 (15)	19 (18)	14	14	16 (16)
300	36 (35)	39 (38)	43 (42)	33	33	39 (37)
500	56 (55)	68 (67)	66 (65)	49	48	59 (57)
1000	111 (111)	121 (121)	115 (114)	89	84	106 (105)

Table 1: Comparison of $ANTCOL$ with five other heuristics

We can first observe that $ANTCOL$ is much better than $Dsatur$. However, both algorithms are based on the same coloring strategy, which is to iteratively select a vertex with a large saturation degree, and to assign to it the best possible color. This means that the ingredients we add to $Dsatur$ in order to elaborate $ANTCOL$ are useful. The same conclusion holds when comparing $Dsatur$ with $ANT\text{-}DSATUR$

Then, we can remark that as soon as $|V| > 100$, $ANTCOL$ is much better than $ANT\text{-}DSATUR$, which is another ant algorithm also based on $Dsatur$. This probably indicates that it is better to make one solution evolve instead of performing several multi-starts. The same kind of remark holds if we compare $ANTCOL$ with $ANT\text{-}RLF$ on graphs with $|V| = 1000$.

Finally, we see that $ANTCOL$ is not competitive at all with $Tabucol$, and thus with GH which uses $Tabucol$ as intensification procedure. We think that this is mainly due to the fact that in $ANTCOL$, too many ingredients are used to define the greedy force and the trail system. Thus, the algorithm is very slow because lots of computation is needed to only change the color of a small number of vertices. This kind of conclusion also holds for the two other existing ant heuristics for the GCP [6], [24].

Remember that in [6], an ant is a constructive heuristic, and in [24], an ant is a local search heuristic. But in contrast, in $ANTCOL$, each individual ant only helps giving a color to a single vertex, which is not a significant role. Hence, we have shown that we can obtain competitive results in comparison with other ant coloring methods by giving a minor role to each ant.

References

1. Bloechliger, I., and Zufferey, N. 2004. A Reactive Tabu Search using Partial Solutions for the Graph Coloring Problem, Technical Report, Institute of Mathematics, Swiss Federal Institute of Technology, Lausanne
2. Brélaz, D. 1979. New Methods to Color Vertices of a Graph, *Communications of ACM* ***22***: 251–256
3. Brown, J.R. 1972. Chromatic Scheduling and the Chromatic Number Problem, *Management Science* ***19***: 456–463
4. Calegari, P., Coray, C., Hertz, A., Kobler, D., and Kuonen, P. 1999. A Taxonomy of Evolutionary Algorithms in Combinatorial Optimization, *Journal of Heuristics* ***5***: 145–158

5. Chams, M., Hertz, A., and de Werra, D. 1987. Some Experiments with Simulated Annealing for Coloring Graphs, *European Journal of Operational Research* ***32***: 260–266
6. Costa, D., and Hertz, A. 1997. Ants can colour graphs, *Journal of the Operational Research Society* ***48***: 295–305
7. Costa, D., Hertz, A., and Dubuis, O. 1995. Embedding of a Sequential Algorithm within an Evolutionary Algorithm for Coloring Problems in Graphs, *Journal of Heuristics 1* ***1***: 105–128
8. Dorigo, M. 1992. Optimization, learning and natural algorithms (in Italian), Unpublished doctoral dissertation, Politecnico di Milano, Dipartimento di Elettronica, Italy
9. Dorigo, M., Di Caro, G., and Gambardella, L.M. 1999. Ant algorithms for discrete optimization, *Artificial Life* ***5***: 137–172
10. Dorigo, M., and Gambardella, L.M. 1997. Ant Colony System: a Cooperative Learning Approach to the Traveling Salesman Problem, *IEEE Transactions on Evolutionary Computation 1* ***1***: 53–66
11. Dorigo, M. Maniezzo, V., and Colorni A. 1991. Positive feedback as a search strategy, Technical Report 91-016, Politecnico di Milano, Dipartimento di Elettronica, Italy
12. Dorigo, M. Maniezzo, V., and Colorni A. 1996. The Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man, and Cybernetics 26* ***1***: 1–13
13. Fleurent, C., and Ferland, J. A. 1996. Genetic and Hybrid Algorithms for Graph Coloring, *Annals of Operations Research* ***63***: 437–461
14. Gamst, A. and Rave, W. 1992. On the frequency assignment in mobile automatic telephone systems, Proceedings of GLOBECOM'92
15. Garey, M., and Johnson, D. 1979. Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York
16. Galinier, P., and Hao, J.K. 1999. Hybrid Evolutionary Algorithms for Graph Coloring, *Journal of Combinatorial Optimization* ***3***: 379–397
17. Galinier, P., and Hertz, A. 2006. A Survey of Local Seach Methods for Graph Coloring, *Computers & Operations Research*, to appear.
18. Herrmann, F., and Hertz, A. 2002. Finding the chromatic number by means of critical graphs, *ACM Journal of Experimental Algorithmics* ***7/10***: 1–9
19. Hertz, A., and de Werra, D. 1987. Using Tabu Search Techniques for Graph Coloring, *Computing* ***39***: 345–351
20. Johnson, D. S, Aragon, C. R., McGeoch, L. A., and Schevon, C. 1991. Optimization by Simulated Annealing: An Experimental Evaluation, Part II; Graph Coloring and Number Partitioning, *Operations Research* ***39***: 378–406
21. Johnson, D. S., and Trick, M. A. 1996. Proceedings of the 2nd DIMACS implementation challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences, *American Mathematical Society* ***26***
22. Leighton, F. T. 1979. A graph coloring algorithm for large scheduling problems, *Journal of Research of the National Bureau Standard* ***84***: 489–505
23. Peemöller, J. 1983. A Correction to Brélaz's Modification of Brown's Coloring Algorithm, *Communications of ACM* ***26***: 593–597
24. Shawe-Taylor, J., Zerovnik, J. 2002. Ants and Graph Coloring, Proceedings of ICANNGA'01: 593–597
25. Stecke, K. 1985. Design planning, scheduling and control problems of flexible manufacturing, *Annals of Operations Research* ***3***: 3–12
26. Stuetzle, T., and Hoos, H. 1997. Improving the Ant System: a Detailed Report on the Max-Min Ant System, Technical Report, Department of Computer Sciences - Intellectics Group, Technical University of Darmstadt
27. Zufferey, N. 2002. Heuristiques pour les Problèmes de la Coloration des Sommets d'un Graphe et d'Affectation de Fréquences avec Polarités, PhD. Thesis, École Polytechnique Fédérale de Lausanne, Switzerland

Variance reduction techniques in the Monte Carlo simulation of clinical electron linear accelerators driven by the ant colony method

***Salvador García-Pareja*[1], *Manuel Vilches*[2], *and Antonio M. Lallena*[3]**

[1] Servicio de Radiofísica Hospitalaría, Hospital Regional Universitario "Carlos Haya", Avda. Carlos Haya, s/n, E-29010 Málaga, Spain.
[2] Servicio de Física y Protección Radiológica, Hospital Regional Universitario "Virgen de las Nieves", Avda. de las Fuerzas Armadas, 2, E-18014 Granada, Spain.
[3] Departamento de Física Moderna, Universidad de Granada, E-18071 Granada, Spain.

Abstract. *The ant colony method is used to drive the application of variance reduction techniques to the simulation of clinical electron linear accelerators of use in cancer therapy. In particular, splitting and Russian roulette, two standard variance reduction methods, are considered. The approach can be applied to any accelerator in a straightforward way and permits, in addition, to investigate the "hot" regions of the accelerator, an information which is basic to develop a source model for this therapy tool.*

1 Introduction

Electron beams produced by linear accelerators (LINAC) are used extensively in radiotherapy treatments. The finite range of the electrons in the tissue allows to treat superficial cancers, avoiding surrounding tissue radiation-induced complications. However, high accuracy dose distributions are essential for assessing the effectiveness of a given treatment. Nowadays, Monte Carlo (MC) simulation is generally considered to be the most accurate approach to electron dose calculation. In order to achieve this accuracy, a detailed information about the characteristics of the electron beam is required. This information includes the energy, angular and spatial distributions of the particles in the clinical beam and to obtain it, a detailed knowledge of the geometrical aspects of the LINAC is needed to simulate the transport of particles through the treatment head. In the MC simulation, the passage of the electrons through the beam modifiers included in the LINAC is followed accounting for all physical processes of clinical significance.

To obtain the required accuracy in the dose calculations, the details concerning the beam are usually determined by means of an iterative process, in which the simulation results are tuned with experimental data until a good match is obtained. Nevertheless, the full MC simulation remains, in general, incompatible with a possible clinical treatment implementation, mainly due to the high computing time required to achieve an acceptable statistical uncertainty.

Computing time can be reduced by recording the phase-space information for each particle that escapes the treatment head. This information includes charge, energy, direction and position of the particle, as well as a tag recording detailed information about its history [1]. Once the phase-space has been obtained, it can be used to complete the simulation on the patient in a second step. A first (minor) problem is linked to the storage requirement for the phase-space files containing millions of photons and electrons [2]. A more important point relies on the fact that, since the beam characteristics are usually different, even for the same type of LINAC, it is necessary to tune each individual LINAC to obtain the phase-space data needed for MC treatment planning. Besides, and depending on the way the simulation is performed, one should analyze each beam configuration individually. Another problem is linked to the correlations that can be generated by the re-use of the phase-space in case more statistics is required to reduce the variance. All these aspects make the quality assurance a difficult task which requires both MC simulation experience and much more time than that usually employed for commissioning a conventional electron treatment planning system.

An alternative to the phase-space approach is the so-called source model [2–6]. The source model considers multiple virtual sources emulating the effect of the LINAC head components in the beam. The tuning process for every individual LINAC consists in adjusting the source parameters. As a result, the dosimetric quantities obtained have a precision comparable to that obtained within the full MC simulation and one avoids storage problems, does not need to simulate the full LINAC geometry (where a major part of the computer time is elapsed) and can safely reduce the uncertainty by increasing the number of electron histories[1] followed.

Variance reduction techniques can be used to make calculations more efficient, whatever the methodology to perform the simulations one chooses [7]. In the case of the simulation of LINACs, where the interest is focused on a localised spatial region (the phantom or the patient), the so-called Splitting (Sp) and Russian Roulette (Rr) techniques appear to be particularly appropriate. The basic idea of these two methods is to favour the flux of radiation towards the region of interest and inhibit the radiation that leaves that region. The variance reduction is accomplished by modifying the relative weights of the particles. Sp should be applied to a particle which "approaches" the region of interest: the particle is transformed in a certain number of identical particles with smaller weights. On the contrary, Rr "kills", with a certain probability, those particles which tend to move away from the region of interest, increasing their weight if they survive. The important point is that when they are used together the simulation remains unbiased [7,8]. The application of these techniques requires a very careful analysis of each particular problem, the most delicate question being how to fix the criteria for switching on each of them. In particular, for the LINAC problem, a lot of simulations must be done to investigate the geometry elements where particles stay for a longer time and to obtain the probabilities that particles coming from the different regions arrive, finally, to the region of interest.

In this paper we discuss how the ant colony method can be used to drive these two variance reduction techniques in MC simulations of LINAC electron beams. The main purpose of this work is to evaluate if the ant colony permits to establish the application criteria of Sp and Rr, avoiding the cumbersome preliminary study above discussed. The results quoted here are for the LINAC Siemens Mevatron KDS and the MC code PENELOPE, although the methodology can be applied to any LINAC and MC code. In the next section the LINAC

[1] The term history corresponds to the simulation of a primary particle and all the secondaries produced by it until their energy falls below the energy cut or leave the system.

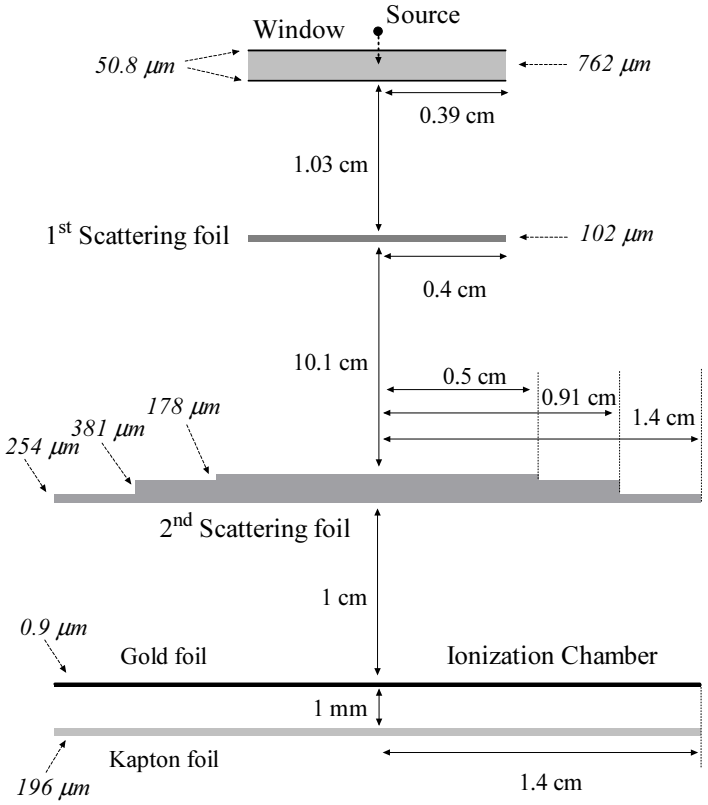


Fig. 1. Scheme of the dispersive elements of the LINAC head.

geometry, the MC code, the variance reduction techniques as well as the way the ant colony method has been implemented are described. In Sect. 3 we compare the results obtained with and without applying the variance reduction techniques. In particular we analyze the influence of these methods with respect to the dose, the computing time and the statistical fluctuations. We draw our conclusions in the last section.

2 Materials and Methods

2.1 The Siemens Mevatron KDS accelerator

This study has been carried out on a Mevatron KDS LINAC manufactured by Siemens. This LINAC permits to use electron beams with nominal energies of 6, 8, 10, 12, 15 and 18 MeV. For the sake of simplicity, in what follows we quote the results obtained for 12 MeV electron beams.

The detailed description of the geometry that is required for the accurate simulation has been provided by the manufacturer. The primary electron beam was assumed to originate

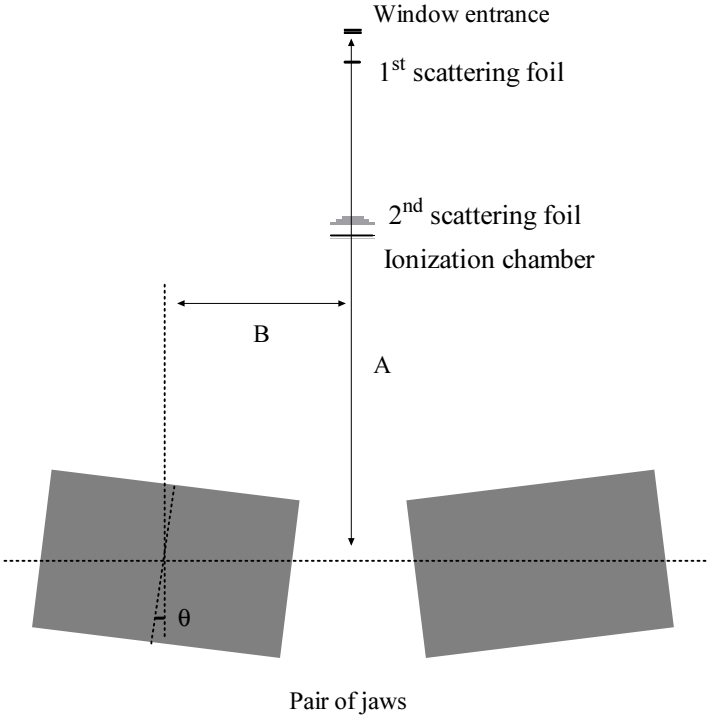


Fig. 2. Scheme of the situation of the jaws.

from a monoenergetic and monodirectional point source (see Fig. 1). Electrons are injected in the treatment head through its entrance window, consisting of two titanium foils of 50.8 μm of thickness with an inner water flux 0.762 mm thick which is needed for cooling. This window has a radius of 3.9 mm. The first scattering foil is situated at 1.03 cm below the entrance window and has a composition and a thickness which depends on the nominal energy selected. For the 12 MeV beam it consists of 102 μm of gold. A second scattering foil made of aluminium permits to achieve the homogeneity of the electron field. It is situated at 10.1 cm of the first scattering foil and consists of three slabs of 178, 381 and 254 μm of thickness and radii 5.0, 9.1 and 14.0 mm, respectively. A cylindrical monitoring ionization chamber of 1.4 cm of radius appears 1 cm below the second scattering foil. It consists of Kapton®[2] and gold foils separating two volumes of air. In actual simulations, the geometry of this chamber is simplified as a couple of slabs (0.9 μm of gold and 196 μm of Kapton®) separated by 1 mm of air.

Two pairs of tungsten jaws (see Fig. 2) limit the radiation field in the directions perpendicular to the beam line (which is chosen to be z). In electron beam mode[3] the jaws keep a fixed position that determine a 25×25 cm^2 field at 100 cm from the source. The upper

[2] Kapton® is a type of polyimide polymer.
[3] The LINAC can be also used to generate photon beams.

(lower) jaws have dimensions of 13.5×11.5×7.5 cm^3 (16.5×11.5×7.5 cm^3), their geometrical centers are situated at A=25.6 cm (33.4 cm) from the entrance window and are shifted a distance B=9 cm in the x direction (10 cm in the y direction). The collimator angle θ is 7.125^o for all jaws.

In order to eliminate the scattered radiation due to the electron scattering in air, an applicator near the patient surface is necessary for the treatment. In our analysis, this element has not been considered because it is a piece strongly dependent on the patient and the particular treatment plan.

Finally, a water phantom is included to determine the depth dose distribution. In the MC simulations done in this work, we have focused on the dose in the beam axis and we have scored the dose in cylindrical voxels with 1 cm of radius and 1 mm of height.

2.2 The PENELOPE code system

The Monte Carlo code system PENELOPE (v. 2003) [8] has been used to simulate the coupled electron/photon transport. It consists of a Fortran 77 subroutine package that handles the transport process as well as the geometry of the problem. The user, in turn, has to provide a main program that controls the evolution of the generated histories and keeps score of the relevant quantities.

According to Berger's terminology [9] and regarding the simulation of the interactions of electrons (and positrons), PENELOPE may be classified as a class II code. Hard collisions (those leading to large angular deflections and/or energy losses) are simulated in a detailed manner, whereas soft events are described by means of a multiple scattering approach. This mixed simulation scheme is implemented through the so-called random-hinge algorithm, which has been proved to be very stable even in extreme situations [10]. On the other hand, photon transport is performed by a conventional detailed simulation procedure.

The speed and accuracy of the simulation of electrons and positrons is determined by the values of the simulation parameters. All simulations reported below, were carried out with "safe" parameters [8, 11]: absorption energies $E_{\rm abs}$ equal to 100 keV and 10 keV for electrons (or positrons) and photons, respectively; mixed simulation cutoffs $W_{\rm cc}$ and $W_{\rm cr}$ equal to 5 keV and 1 keV, respectively; parameters C_1 and C_2, which determine the angular deflection cut-offs, equal to 0.05. The maximum allowed step length s_{max} was fixed to one tenth of the characteristic thickness of each body.

2.3 Variance reduction techniques

In analogue MC calculations in which the full stochastic transport of particles is simulated, the trivial way to reduce the estimated variance is to run more histories until the desired level of accuracy is reached. The reduction factor is proportional to the squared root of the number of histories followed.

In MC simulations is a common practice to define the efficiency, ϵ, of a given procedure

$$\epsilon = \frac{1}{s^2 T}\,, \tag{1}$$

where s^2 is the estimated variance and T is the computing time used. According to Eq. (1), the efficiency of MC simulations can be increased by reducing the variance or reducing the

time per history followed. The techniques leading to an increase in the efficiency, even if they are not related to variance reduction, are usually called variance reduction techniques.

For the problem we are interested in, that is the simulation of a LINAC, the important quantity is the dose deposited in the water phantom. Then, a way to reduce the variance is to follow completely those histories which are expected to give contribution to this dose and to forget the remaining ones. The discrimination between both types of histories is usually done by taking care of the regions of the geometry they go through. There are two techniques which are very useful in this respect when used together: Sp and Rr [7,8]. The basic idea of these methods is to favour the flux of radiation towards the region of interest and inhibit the radiation that leaves that region. Variance reduction is accomplished by modifying the weights of the particles simulated. It is assumed that primary particles start moving with unit weight. On the other hand, the initial weight of any produced particle equals that of the particle which produced it. Sp consists in transforming a particle, with weight w_0 and in a certain state, into a number $s > 1$ of identical particles with weights $w = w_0/s$ in the same state. Sp should be applied when the particle approaches the region of interest. The Rr technique is related to the reverse process: when a particle tends to move away from the region of interest it is killed with a certain probability, $r < 1$, and, if it survives, its weight is increased by a factor $1/(1-r)$. In this way, splitting and killing leave the simulation unbiased. After completing a number N of histories, the expected value $\langle F \rangle$ of a certain quantity of interest F is estimated by means of the average

$$\overline{F} = \frac{1}{N} \sum_{i,j} w_{ij} f_{ij} \,, \tag{2}$$

where w_{ij} and f_{ij} denote, respectively, the weight and the contribution to the score of the j-th particle (j=1 for the primary and $j > 1$ for the secondary) of the i-th history. The central limit theorem ensures that, when $N \to \infty$, the probability distribution of F is Gaussian and then, in this limit,

$$s^2 = \frac{1}{N} \left[\frac{1}{N} \sum_{i} \left(\sum_{j} w_{ij} f_{ij} \right)^2 - \overline{F}^2 \right] \tag{3}$$

is an unbiased estimator of the variance of $\overline{F}$. As every scoring voxel has its own uncertainty, in this work we have used the variance of the voxel with maximum dose to estimate the efficiency of the simulation. Simulation results are commonly expressed in the form $\overline{F} \pm ks$. All the uncertainties quoted throughout this paper have been obtained using $k = 3$, so that the probability that the true value F lies within the error bar is 99.7%.

The efficiency of the Rr and the Sp relies on the strategy used to decide when splitting and killing must to be applied. If Rr is used in such a way that many particles surviving with high weight reach the region of interest, the variance increases and the efficiency reduces. The same occurs if Sp is used in regions were very few particles passing through them contribute to the quantity of interest, because too much time is occupied simulating histories with low contribution.

In our problem, the beam is mainly intercepted by the jaws. For example, the angular distribution of the electrons of a 12 MeV beam after passing through the scattering foils, considering the full geometry of the LINAC, is shown in Fig. 3. The jaws intercept electrons with polar angles larger than 7.125° and, as we can see, a large fraction of the initial electrons can be absorbed. In fact, the percentage of primary electrons reaching the water phantom

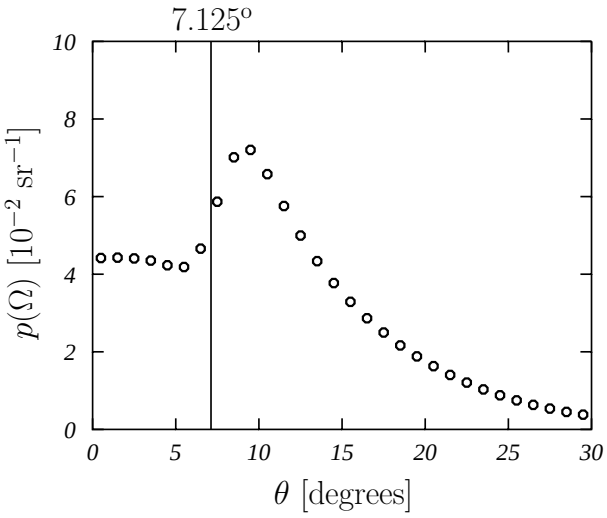


Fig. 3. Angular distribution, per solid angle, for the 12 MeV beam below the scattering foils. Electrons with angles greater than 7.125° are intercepted by jaws. Error bars are smaller than the symbols used.

for the 12 MeV beam is $17.29 \pm 0.02\%$. Similar values are obtained for other energies. However, we can not neglect completely the electrons absorbed by the jaws because they can contribute to the dose in the phantom by means of the secondary particles they generate, mainly Bremsstrahlung photons. Then, a lot of computing time must be expend to simulate these particles which give a rather small contribution. This is a nice situation to apply Rr. The increase in the variance produced by the higher weight of the surviving secondary particles is then rearranged introducing Sp for those particles which approach the phantom.

The important point is how and where Rr and Sp must be applied. In a previous work [12], some "ad hoc" criteria were adopted. As said above, a major part of the simulation time is elapsed by particles inside the jaws. We applied Rr to particles entering the jaws at a distance from their inner side larger than a value which is related to the range of the more energetic particles moving inside the jaws. For the case we are analyzing here, this distance was fixed to 4 mm and a probability $r = 0.9$ was chosen. Additionally, and for each body in the geometry, a threshold energy was fixed in such a way that those particles with an energy smaller than these cut values were killed with a certain probability. The threshold energy was fixed to 3 MeV and the probabilities ranged from $r = 0.7$ (for the phantom and the air above it) and 0.9 (for the entrance window, the dispersive foils and the ionization chamber). Sp was applied to particles reaching the phantom with $s = 50$. The combined use of Sp and Rr permitted to reduce the simulation time by a factor ~ 40.

2.4 Ant colony approach

The CPU time reduction just mentioned implies a noticeable improvement in the LINAC simulation. However, the approach described requires a careful analysis of the geometry of the system and needs of an important number of simulations allowing to get a detailed knowledge about which one of regions in the simulation space are more expensive (in the

sense of CPU time). It is evident that this procedure cannot be blindly applied to other LINACs, being necessary to develop the same kind of study for each system that one wants to simulate.

In this work we propose to use a method based on an ant colony algorithm to drive the application of Rr and Sp. The purpose is to have a procedure, robust enough to be applied to any LINAC without previous analysis.

Ant colony algorithms were first introduced by Dorigo *et al.* [13]. These algorithms are inspired in the behavior of actual ant colonies and permit to create optimization tools to solve problems like that of the traveling salesman [14]. The problem is formulated in terms of ants going out from the nest to look for food. In their trajectories ants leave pheromone and the level of pheromone in a trajectory increases as a function of the number of ants which travel by this trajectory, while it is reduced when it is not visited with a certain frequency. The key point is to define a parameter which plays the role of the level of pheromone. In our case the analogy established is that electrons go from the source (the nest) to the detection region (the food). If we force electrons to follow a particular trail, our results would show a clear bias, but playing Rr when electrons loose the trail and Sp when they follow it, leave the simulation unbiased.

We have divided the full geometry of our problem in a set of fictitious volume cells which are characterized by a value of a parameter known as *importance*, I. This parameter is the analogue of the pheromone level and formally is a measure of the expected contribution of the particles which reach a given cell and score to the quantity interest. In its simplest form, the logic is to split particles when they pass from a cell with a given value of I to another with a higher one. Rr is applied when the importance reduces. The ratio s/r is governed by the ratio of the importance on both sides of the boundary. A similar method was implemented in the MC code MCBEND [15, 16] to solve shielding problems.

The importance map can have as much dimensions as needed because one can include not only spatial characteristics, but also energy, direction of motion, particle type, material, etc. In our case, we have considered three spatial dimensions (cubes sided 1 cm), two energy cells, dividing the initial electron energy in two, and each cell is divided additionally in two, according to the material (air or any other scatterer) filling it. The importance map is only defined for electrons whose simulation is more time consuming.

To define the importance values in each cell a dynamic process is considered. Starting the simulation, the complete map has value I equal to 1. The weights of particles take also value 1 at the source. During the simulation two quantities are scored. The first is $N_e^{\mathrm{B}}(i)$, which is the number of electrons beginning a step of their track in i-th cell. The second is $N_e^{\mathrm{D}}(i)$, which is the number of electrons that, having been scored in $N_e^{\mathrm{B}}(i)$, reach the detection region. Thus, the ratio of electrons passing through the i-th cell and reaching the region of interest is

$$P(i) = \frac{N_e^{\mathrm{D}}(i)}{N_e^{\mathrm{B}}(i)}\,. \tag{4}$$

Obviously, $P(i)$ can only take values from 0 (no electrons passing trough that cell reach the detector) to 1 (all electrons reach it). This magnitude help us to define the importance $I(i)$ of a given cell. In principle, one can use any increasing function with a minimum when $P(i) = 0$ and a maximum when $P(i) = 1$. It is worth to take care of the values of the importance because if the criterion chosen is not appropriate, particles with very different weights could arrive to the region of interest and the uncertainty of the simulated quantities

will increase. In our case we have assumed

$$I(i) = 2^k \,, \tag{5}$$

with

$$k = \begin{cases} \left[5\dfrac{P(i)}{P(0)} - 5\right], & \text{if } \; P(i) \leq P(0) \\ \left[7\dfrac{P(i) - P(0)}{1 - P(0)}\right], & \text{if } \; P(i) > P(0) \end{cases} . \tag{6}$$

Here $[\alpha]$ indicates the integer part of α and $P(0)$ the ratio of electrons that starting from the source reach the detection region. In this manner, I takes values from 2^{-5} to 2^7 and $I(0)$ is always equal to 1. These limit values are arbitrary but have to be tested because if the interval is too broad or too narrow the efficiency could decrease. $P(i)$ changes during the simulation and $I(i)$ also do it.

If the particle which is simulated has a weight w and is at the i-th cell, the algorithm that uses Rr or Sp is as follows:

- If $I(i) \cdot w > 1$, split the particle in $s = I(i) \cdot w$ particles, each one with corrected weight $w' = w/s = [I(i)]^{-1}$.
- If $I(i) \cdot w < 1$, apply Rr with probability $r = 1 - I(i) \cdot w$; if the particle survives, the new weight is $w' = w/(1 - r) = [I(i)]^{-1}$.
- If $I(i) \cdot w = 1$, neither Sp nor Rr is applied.

The continuous redefinition of the value of the importance in each cell (each history modifies the value of I in the cells it pass trough) points out the cooperative characteristic of this procedure.

3 Results

Following the methodology described in the previous section, we run two kinds of simulations: (i) analogue simulations with no variance reduction and (ii) accelerated simulations using the importance map with Rr and Sp. For the results quoted below, analogue simulations were done by following $4 \cdot 10^7$ histories, while accelerated simulations were performed following $7 \cdot 10^6$ histories. Calculations have been done with a PC Pentium 4 at 1.6 GHz, with 512 MB of RAM memory and Windows XP operating system. The Intel Fortran Compiler 7.0 with the compilation option of the velocity optimization was used.

Fig. 4 shows a comparison between the depth dose distributions obtained for the 12 MeV electron beam with both types of simulations. Therein, the solid line represents the results calculated using the importance map driving the Sp and the Rr methods. Symbols represent the results found with the analogue simulation, without variance reduction techniques. The uncertainties of the results obtained within the accelerated scheme are similar to those quoted for the analogue simulation.

As we can see the agreement between both calculations is remarkable and this points out the feasibility and validity of the methodology proposed. The advantage of this last is evident in Table 1, where the CPU time elapsed in the complete simulations, the value of s for the voxel with maximum depth dose and the efficiency are compared. As we can see,

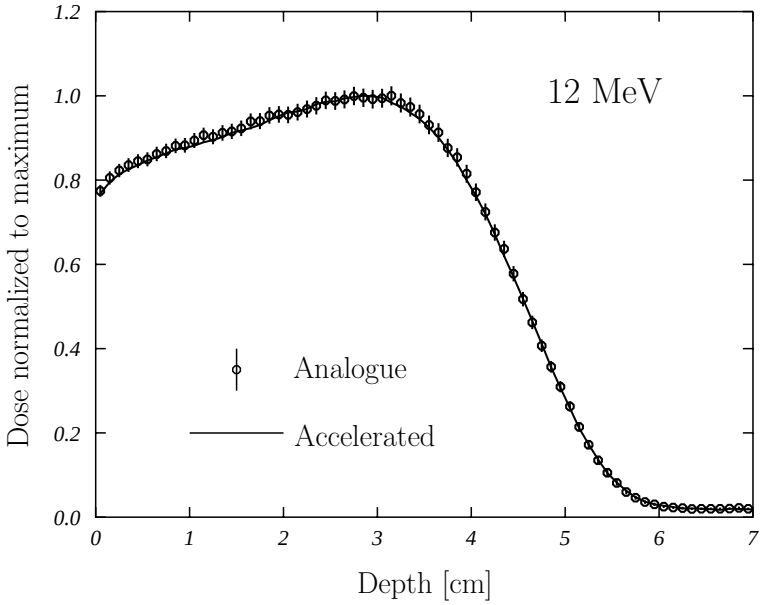


Fig. 4. Depth dose distribution in the beam axis as a function of the depth in the water phantom for the 12 MeV electron beam. Symbols represent the accelerated calculation. Solid line corresponds to the analogue simulation. Error bars in this last case are of the same order as in the accelerated simulation.

Simulation	CPU time	$s_{\max}$ [%]	ϵ [s^{-1}]
Analogue	220.0 h	2.0	3.04×10^{-3}
Accelerated	5.3 h	1.8	3.58×10^{-1}

Table 1. CPU time, standard deviation for maximum dose, and efficiency for the 12 MeV electron beam.

a similar level of uncertainty is found in both calculations ($s_{\max} = 2.0\%$ in the analogue simulations and 1.8% in the accelerated one), but the CPU time is reduced by a factor larger than 40 and the efficiency is increased by two order of magnitude.

To finish, we show in Fig. 5 the importance map obtained in our calculations for the high energy particles (that is electrons with energy larger than half the initial energy) in air (left) and for low energy electrons in materials different to air (right). Therein black regions correspond to cells where no particles pass through. White regions are those showing a higher value of the importance parameter. In the right panel one can distinguish the scattering foils and the jaws. Also the phantom is evident. This map inform us about those regions of the geometry which act as radiation sources for the particular situation which is being analyzed. This information is strongly important if one is trying to develop a source model for the LINAC.

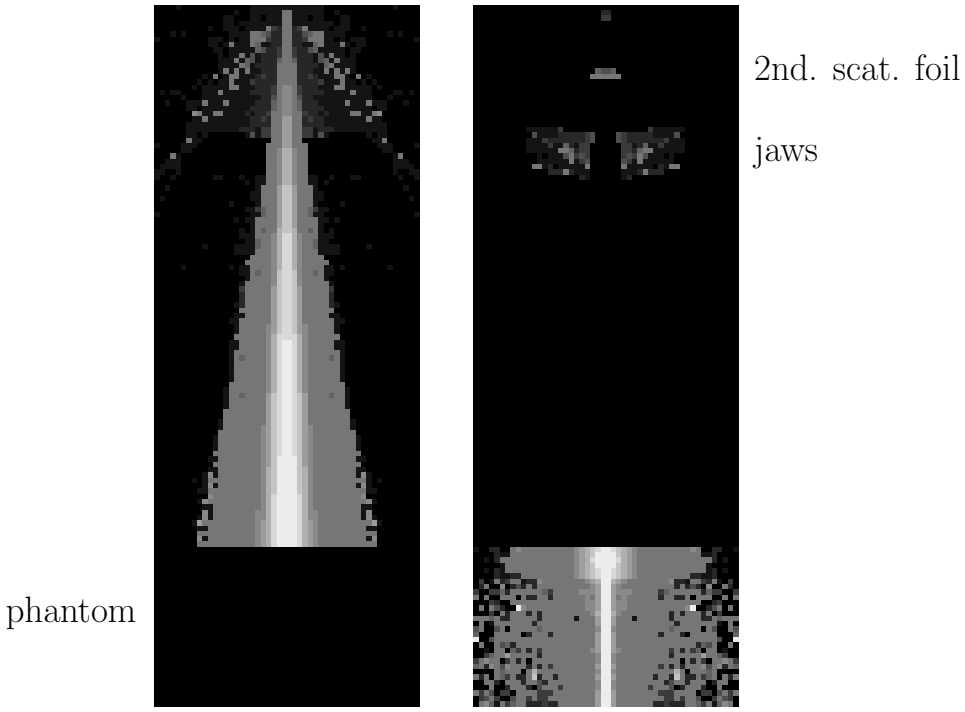


Fig. 5. Importance map for high energy electrons in air (left) and low energy electrons in materials different to air (right).

4 Conclusions

In this work we have used the ant colony method to drive the application of two standard variance reduction techniques, Sp and Rr, which are used together in practice. An application to the simulation of a LINAC utilized in cancer therapy has been developed.

Results show a very good agreement between the simulations performed by applying the variance reduction techniques and those done in a full analogue way, with no acceleration at all. CPU times are reduced by a factor 40, whereas the efficiency of the simulation increases by two orders of magnitude.

Finally, the importance map permits to obtain information about the actual radiation sources of the LINAC and contributes to an easier development of a source model of it.

The use of the ant colony approach to drive Rr and Sp variance reduction methods, does not improve the CPU time needed for the simulation in comparison with the application of Rr and Sp methods based on "ad hoc" criteria fixed for the particular LINAC studied. However, the procedure can be applied to any LINAC and for any MC simulation code without the previous detailed study of the geometry of the system which is mandatory to establish the right Rr and Sp application criteria.

References

1. D.W. Rogers, B.A. Faddegon, G.X. Ding, C.-M. Ma and J. We, "BEAM: A Monte Carlo code to simulate radiotherapy treatment units," Med. Phys. **22**, 503-525 (1995).
2. C.-M. Ma, "Accurate characterization of Monte Carlo calculated electron beams for radiotherapy," Med. Phys. **24**, 401-416 (1997).
3. C.-M. Ma and D.W. Rogers, "Beam characterization: a multiple source model," National Research Council of Canada Report PIRS-0509D (NRC, Ottawa, 1995).
4. C.-M. Ma, "Characterization of computer simulated radiotherapy beams for Monte Carlo treatment planning," Radiat. Phys. Chem. **53**, 329-344 (1998).
5. S.B. Jiang, A. Kapur and C.-M. Ma, "Electron beam modeling and commissioning for Monte Carlo treatment planning," Med. Phys. **27**, 180-191 (2000).
6. A. Trindade, P. Rodrigues, L. Peralta, M.C. Lopes, C. Alves and A. Chaves, "Fast electron beam simulation and dose calculation in radiotherapy," Nucl. Instr. Meth. Phys. Res. A **522**, 568-578 (2004).
7. A.F. Bielajew, "Fundamentals of the Monte Carlo method for neutral and charged particle transport, " (The University of Michigan, Ann Arbor, 2001).
8. F. Salvat, J.M. Fernández-Varea and J. Sempau, "PENELOPE—A Code System for Monte Carlo Simulation of Electron and Photon Transport," (OECD Nuclear Energy Agency, Paris, 2003).
9. J.M. Berger, "Monte Carlo calculation of the penetration and diffusion of fast charged particles," in Methods in Computational Physics vol 1, ed. B. Alder, S. Fernbach and M. Rotenberg (Academic, New York, 1963).
10. A.F. Bielajew and F. Salvat, "Improved electron transport mechanics in the PENELOPE Monte-Carlo model," Nucl. Instrum. Meth. Phys. Res. B **173** 332-343 (2001).
11. J. Sempau, A. Sánchez-Reyes, F. Salvat, H. Oulad Ben Tahar, S.B. Jiang and J.M. Fernández-Varea, "Monte Carlo Simulation of Electron Beams from an Accelerator Head Using PENELOPE," Phys. Med. Biol., **46**, 1163-1186 (2001).
12. S. García Pareja, "Análisis físico del acelerador lineal de electrones Siemens Mevatron KDS con PENELOPE," Master Thesis, Universidad de Granada, 2004 (unpublished).
13. M. Dorigo, V. Maniezzo and A. Colorni "Ant System: Optimization by a Colony of Cooperating Agents," IEEE Transactions on Systems, Man., and Cybernetics Part B, **26**, 29-41 (1996).
14. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy-Kan and D.B. Shmoys "The traveling salesman problem," (Wiley, New York, 1985).
15. T. Shuttleworth, M. Grimstone and S. Chucas, "Application of Acceleration Techniques in MCBEND," Proc. Ninth Int. Conf. on Radiation Shielding (Tsukuba, 1999) J. Nucl. Sci. Techn., Suppl. 1, 406 (2000).
16. G.A. Wright, T. Shuttleworth, M. Grimstone and A.J. Bird, "The status of the general radiation transport code MCBEND," Nucl. Instr. Meth. Phys. Res. B **213**, 162-166 (2004).

Tradinnova-ACO: Ant Colony Optimization Metaheuristic applied to Stock-Market Investment

Isidoro J. Casanova and José M. Cadenas

Dpto. Ingeniería de la Información y las Comunicaciones.
Facultad de Informática. Universidad de Murcia.
Campus de Espinardo. 30100 Murcia. Spain.
isidoroj@um.es
jcadenas@um.es

Abstract. *A stock market investor buys and sells stocks in order to obtain the best possible profit. This dealing can be depicted on a graph, on which each node represents the stocks purchased. In this way, this paper proposes the use of ant colony optimization for calculating the best sequence when buying and selling, thereby helping when deciding how to invest.*

Key words: Stock-Market Investment, Ant Colony Optimization, Intelligent Systems, Decision Making, Heuristics.

1 Introduction

There are different techniques which allow us to see whether a particular stock is valued as cheap or expensive (fundamental analysis) or whether the price is in an upward or downward trend (technical analysis). However, an investor should not really focus on just the one particular stock, but on all the stocks making up a specific market (in our case the Continuous Spanish Market), or even focus on stocks in other countries, in order to create a stock portfolio.

When creating a portfolio, you can take into account the risk aversion of the investor, who, following the Markowitz theory, will want to obtain maximum profit with minimum risk ([16]). Usually, when a portfolio of stocks is created, the investor hopes to make a profit within a specific period of time, without a dynamic management of the stocks making up the portfolio.

In [1], we proposed a heuristic algorithm called TRADINNOVA, which intelligently simulates the performance of an investor in the Continuous Market applying rules in order to buy and sell stocks. This algorithm is based on the premise that on any given day in the stock market you can select a combination of stocks with which you hope to make a good profit. The heuristic techniques in order to select these stocks can be extremely diverse, from technical or fundamental analysis to artificial intelligence techniques such as neuronal networks, Bayesian networks, news analysis, market climate analysis, etc ([6–13]). Once these stocks have been selected, TRADINNOVA chooses the best stock to buy and its purchase price. As the days go by it decides when to sell and the selling price, what other stocks can be bought and also follows up the buying or selling orders which are not carried out, in order to act upon them.

TRADINNOVA dynamically manages stocks in a period of time, widening the traditional Markowitz model which does not stipulate the moment for buying and selling the stocks, and extending the capacities of selection of the stocks as it is the investor who chooses the heuristic technique to use when selecting stocks, thereby adapting the algorithm to the personal preferences of the investor.

In this paper, we present an extension of TRADINNOVA, called TRADINNOVA-ACO, in which in order to select the stocks to be bought or sold an ant colony optimization algorithm is used.

The paper is organized in the following way: firstly, we introduce the field of stock market investment, commenting on the prediction techniques which are generally used, and we introduce the concept of the optimum sequence in buying and selling. In section 3, we introduce ant colony optimization, seeing how the problem is characterized, in order to then see, in the following section, how this metaheuristic can be used to make stock market predictions. Finally, we finish with conclusions and future research.

2 Stock market investment

2.1 Prediction techniques

Apart from the techniques which try to determine whether a stock is valued as cheap or expensive (fundamental analysis) or whether its price is in an upward or downward trend (technical analysis), there are numerous artificial intelligence studies to predict the value of a specific stock, with neuronal networks being the most frequently used technique ([6–8]). Genetic algorithms are also used to optimize the parameters of these neuronal networks or to optimize the technical analysis of a stock ([9, 10]).

There are also studies into how to predict the performance of a stock from news published on the Internet ([12, 13]).

If we go by [11], the techniques to apply in finance can be divided into 5 categories (Fig. 1).

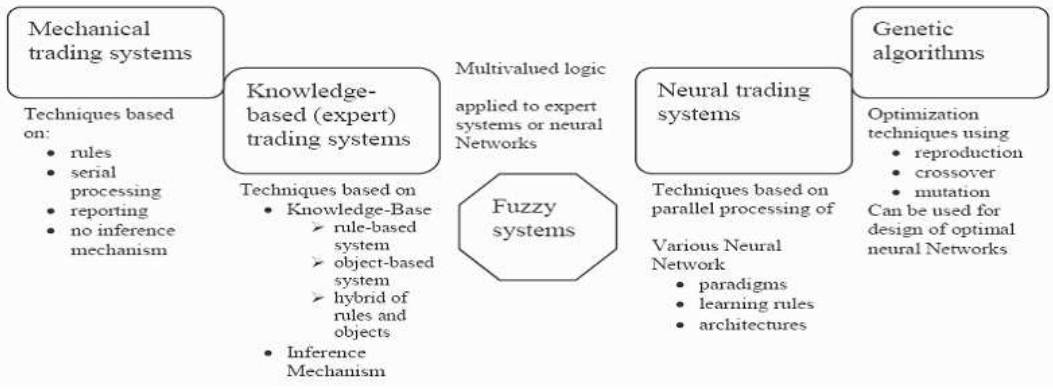


Fig. 1. Classification of the techniques to apply to finance

All techniques advise us on the purchase of a particular stock. However, once the performance of a stock, or a group of stocks, has been predicted, it would be necessary to apply a policy of buying and selling in order to maximize the profit and thereby provide a dynamic management of the stocks, as done in TRADINNOVA.

2.2 Optimum dealing sequence

As the days go by, in the Continuous Market, stocks rise and fall by a certain percentage, and an investor will sell and buy stocks throughout those days. But, what would be the ideal dealing sequence in order to maximize profit?. In principle, the answer would be easy if we already knew how the market was going to perform, as the ideal situation would be to invest, each day, all of the money in the stock in the market which rises the most on that day, as long as the stock rises enough so that commissions don't lose us money in the end.

It is of course, impossible to know which stock this will be. Due to this drawback, what the investor has to do is invest in the stock which has the best chance of rising (statistically, technically, fundamentally, etc). Obviously, a stock has more chance of rising if it is in an upward trend, and supposedly, if a stock is in an upward trend, the purchase of that stock will form part of an "optimum" dealing sequence.

In order to obtain this "optimum" dealing sequence we are going to use the metaheuristic of the ant colony. Here we will briefly describe this metaheuristic.

3 Ant Colony Optimization (ACO)

3.1 Introduction

Ants are social insects which live in colonies and, due to their mutual collaboration, are capable of displaying complex behavior and completing difficult tasks from the point of view of an individual ant.

While they move within the ant hill and the food source, some species of ants deposit a chemical substance known as pheromone (a substance which can be "smelt"). If no trace of pheromone is found, the ants move in a basically random manner, but when deposited pheromone is in existence, they are more likely to follow the trail. Given that ants deposit pheromone on the path they are following, this behavior leads to positive feedback, concluding in the formation of a trail marked out by a higher concentration of pheromone. This behavior allows ants to find the shortest paths between the ant hill and the food source.

3.2 From natural ants to the ant colony optimization metaheuristic

ACO algorithms are directly inspired by the behavior of real ant colonies in order to solve combinatory optimization problems (NP-Hard problems). The artificial ant is a simple computational agent which tries to find possible solutions to the problem by exploiting the available pheromone trails and heuristic information. ACO algorithms are essentially constructive algorithms: in each repetition of the algorithm, each ant builds a solution to the problem referring to a construction graph.

Characterization of the problems. In general the problems to be resolved by this algorithm have the following characteristics:

1. There is a finite set of components $N = \{n_1, n_2, \ldots, n_k\}$, each of these will be represented by a node on a graph.
2. There is a set of constraints $REST$ defined by the problem to be solved.
3. The problem presents diverse states which are defined according to components ordered by sequences $EST =< n_r, n_s, \ldots, n_u, \ldots >$ ($< r, s, \ldots, u, \ldots >$ to simplify) on the elements of N. If T is the set of all possible sequences, we will call S the set of possible (sub)sequences which respect the constraints $REST$. The elements in S define the possible status, which correspond with paths on the graph, that is, sequences of nodes or edges.
4. There is a neighborhood structure which defines the edges of the graph. So a_{rs} would be the connection/transition of n_r with its neighbor of n_s if:
   - Both n_r and n_s belong to T.
   - The state n_s can be reached from n_r in a logical step, there should be a valid transition between r and s. The neighborhood reachable by n_r is the set which contains all of the sequences $N_s \in S$.
5. Explicit costs, c_{rs}, associated with each edge must exist.
6. A solution, SOL, is an element of S which verifies all of the requisites of the problem.
7. There is a cost, $C(SOL)$, associated with each solution SOL.
8. In some cases, it is possible to associate a cost, or an estimation of it can be associated with the states.

Structure of a generic ACO algorithm. Next we will show the original algorithm in pseudocode of the ant colony metaheuristic optimization, as specified in [3].

```
Procedure ACO-Meta_heuristic()
   while (termination_criterion_not_satisfied)
      schedule activities
         ants_generation_and_activity();
         pheromone_evaporation();
         daemon_actions(); {optional}
      end schedule_activities
   end while
end Procedure

Procedure ants_generation_and_activity()
   while (available_resources)
      schedule_the_creation_of_a_new_ant();
      new_active_ant();
   end while
end Procedure
```

```
Procedure new_active_ant() {ant lifecycle}
   initialize_ant();
   M = update_ant_memory();
   while (current state <> target state)
      A = read_local_ant-routing_table();
      P=compute transition probabilities(A,M,problem_constraints);
      next_state=apply_ant_decision_policy(P,problem_constraints);
      move_to_next_state(next_state);
      if (online_step-by-step_pheromone_update)
         deposit_pheromone_on_the_visited_arc();
         update_ant_routing_table();
      end if
      M = update_internal_state();
   end while
   if (online_delayed_pheromone_update)
      evaluate_solution();
      deposit_pheromone_on_all_visited_arcs();
      update_ant-routing_table();
   end if
   die();
end Procedure
```

Fig. 2. Original ACO algorithm

How it works. The ants (artificial) in a colony move, concurrently and asynchronously, through the states adjacent to the problem (which can be represented in graph form with weights). This movement is carried out by following a rule of transition based on the local information available in the components (nodes). This local information includes the heuristic and memoristic information (pheromone trail) to guide the search. On moving around the construction graph, the ants build solutions incrementally. Optionally, the ants can deposit pheromone each time they cross an arc (connection) while they build the solution (linear renewal, step by step, of the pheromone trail). Once each ant has generated a solution it is evaluated and can deposit a quantity of pheromone depending on the quality of the solution (linear renewal of the pheromone trails). This information will guide the search of the other ants in the colony in the future.

4 Use of the ant colony for stock market prediction

4.1 Characterization of the problem

What we will attempt to optimize is the daily sequence of buying and selling stocks, starting from their historic price. Therefore each node on the graph will contain a specific stock next to the price registered on a particular day.

Each day it will be possible to either have the money invested in one of these stocks or keep it in cash, and the next day, keep the stock that we had, or sell it in order to recover the money, with which you may (or may not) wish to acquire a new stock.

This deal will produce a profit which accumulates over the days. To select which stock it is best to invest in each day we can use different heuristic techniques, such as technical stock analysis.

To illustrate the problem, we can observe Fig. 3, in which on the first day the investor has a quantity of money which can be invested (or not) in any of the stocks which constitute the market, in this case, A, B or C. For the following day, the investor can either change where they have the stock or cash them in. The days will continue to go by until money has definitely been made and we can check how much money has finally been made with the dealing sequence. What we are trying to discover is the most profitable buying and selling sequence.

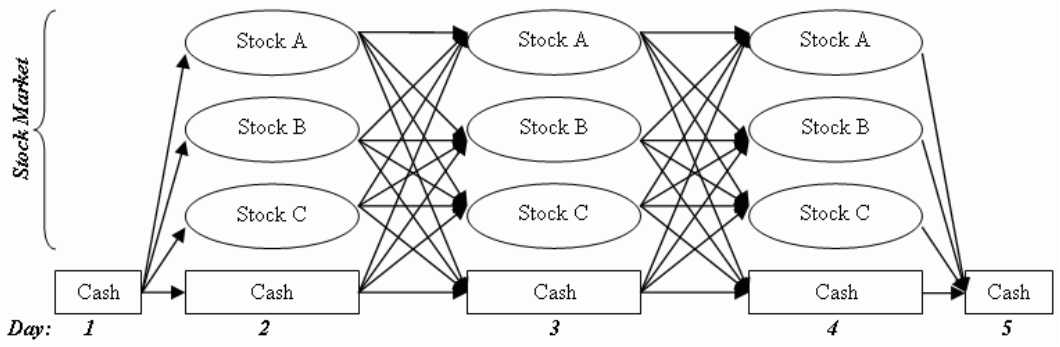


Fig. 3. Graph for stock dealing

If we go by the ACO terminology explained in 3.2 each ant would represent an investor buying and selling stocks on the continuous market:

1. For each day there would be different nodes on the graph representing the stocks and their values: $N = \{a_1, b_1, c_1, \ldots, n_1, a_2, b_2, c_2, \ldots, n_2, a_m,\ b_m, c_m, \ldots, n_m\}$. Where $(a, b, c, \ldots, n)$ represent the prices fixed by n different stocks from day 1 until day m.
2. There are constraints to be applied ($REST$) so if each day an ant is in one of the nodes corresponding to that day, the next day it will either be able to continue in that node or jump to any of the other nodes corresponding to the following day.
3. The states of the problems are any combination of the elements of N. We would designate as S the sets of possible (sub)sequences which respect the constraints $REST$. The elements in S would define the buying and selling done by an investor over m days.
4. The neighbourhood structure which defines the edges of the graph will be defined so that if we find ourselves in the node $i_r \in (a_r, b_r, c_r, \ldots, n_r)$, we will only be able to move to node j_s as long as $i \neq j$ y $s > r$. In other words, if we find ourselves invested in a

stock i on one day r, the transition which would be considered valid would be to invest in a stock j (different to the previous one) on any other day after s.

5. There are costs or profits which are produced as we move around the edges, depending on whether there is a profit or loss made on the different stock deals.
6. A solution SOL is an element of S that verifies all of the requisites of the problem.
7. There is a profit $B(SOL)$ associated with each solution SOL.
8. When a state has been invested in, we can calculate the profit expected by moving to another state using different heuristic techniques (technical, fundamental, neuronal network analysis).

4.2 Application of the metaheuristic: TRADINNOVA-ACO

The proposal is the application of the following ACO metaheuristic algorithm to stock market investment (modified from the original ACO algorithm specified in [2] and [3]):

```
Procedure Tradinnova-ACO()
   parameter-initialization
   while (termination-criterion-not-satisfied)
      schedule-activities
         investor-generation-and-activity()
         pheromone-evaporation()
         daemon-actions() {optional}
      end schedule-activities
   end while
end Procedure

Procedure investor-generation-and-activity()
   repeat in parallel for k=1 to m (number-of-investors)
      new-investor(k)
   end repeat in parallel
end Procedure

Procedure new-investor(investor-id)
   initialize-investor(investor-id)
   BS = initialize-investor-buy-sell-memory()
   while (current-date <> end-date)
      STC = compute-recommended-stocks ( Heuristic-Type, BS )
      selected-stock = apply-decision-policy ( STC, BS)
      buy-and-sell-to-move-to-next-state( selected-stock )
      if (on-line-step-by-step-pheromone-update)
         deposit-pheromone-on-the-visited-edge()
      end if
      BS = update-buy-sell-internal-state()
   end while
```

```
   if (online-delayed-pheromone-update)
      for each visited edge
         deposit-pheromone-on-the-visited-edge()
      end for
   end if
   release-investor-resources(investor-id)
end Procedure
```

Fig. 4. TRADINNOVA-ACO: Application of the ACO metaheuristic to stock-market investment

In the proposed algorithm, instead of ants there are investors who will take different investment paths, while dealing, from an initial to a final date. The first step includes the initializing of the values of the parameters which are taken into consideration in the algorithm. Amongst others, this means fixing the first pheromone trail associated with each transition, the number of investors in the market (m), and the weights defining the proportion in which they will affect the heuristic and memoristic information in the rule of probabilistic transition.

The main procedure of TRADINNOVA-ACO controls, through the constructor *schedule-activities*, the planning of the following three components:

- generating and bringing into operation the investors,
- the evaporation of pheromone, and
- the daemon actions.

On the other hand, the *initialize-investor-buy-sell-memory()* procedure takes care of specifying which stock is bought initially, as it is considered as the initial state at which the investor begins his or her path as well as storing the corresponding component in the memory of the investor BS (the buying-selling sequence to be carried out).

Finally, we must mention that the *compute-recommended-stocks* and *apply-decision-policy* procedures take into consideration the current state of the investor, the current values of the pheromone visible in the said node and the constraints of the problem, in order to establish the probabilistic transition process towards other valid states.

4.3 Making the prediction

The supposition on which we calculate the optimum stock dealing sequence, in other words, that which produces the greatest profit, is that we will be able to focus on where we have invested on the last day, with there being two possible cases: that the money is either invested in a specific stock or cashed in.

If the money is cashed in, this will be due to the fact that it is not currently of interest to invest in a stock, but if the money is invested in a particular stock, we can assume that this latest stock chosen is in an upward trend, meaning that if we maintain this investment, we will achieve the greatest profits from investing in the best possible stock.

In other words, our algorithm proposes buying the last stock found within the optimum stock buying-selling sequence.

5 Preliminary studies and experiments

We have tested TRADINNOVA making a simulation in the Spanish Continuous Market since years 2001 and 2004, having in account operating of the market, applying commissions and giving the orders of buying and selling limited to a price. In the simulation, the most representative stocks (between that they had official quotation from the 02 January 2001 to the 30 December 2004) of the Spanish continuous market are considering. Of the IBEX35, 28 stocks have been chosen and 9 stocks of the remainder of the market. By means of a file .INI the different parameters of input to the algorithm are specified.

The experiments made with TRADINNOVA and the definition of the parameters are shown in [1].

The obtained results have been made with a simulation in different periods, to have more empirical data, obtaining in general quite good results. In the Fig. 5 we show, for the different periods in which the simulation has been made, the revaluation obtained by the IBEX35 and with TRADINNOVA.

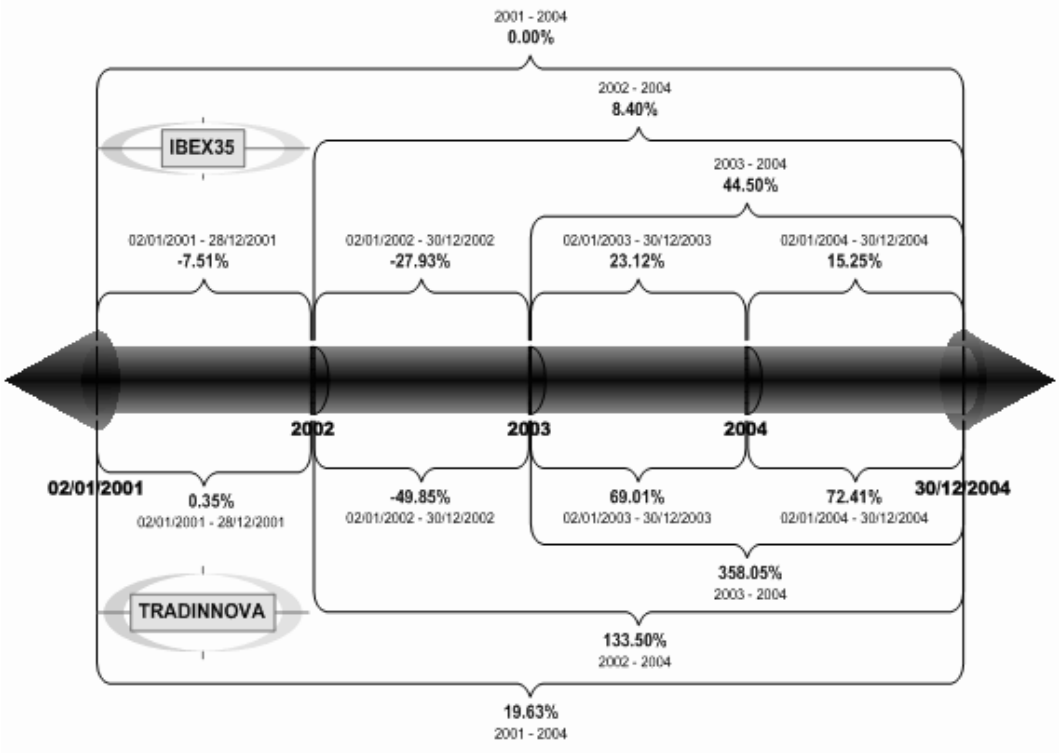


Fig. 5. IBEX35 vs TRADINNOVA

As can be seen in [1], the results obtained by this algorithm are extremely encouraging, but a pending subject is to improve the criteria for selection. With TRADINNOVA-ACO we propose a new criteria to select the stocks through the optimum sequence of buying and selling stocks.

This subject is the objective of the TRADINNOVA-ACO algorithm. We are in the phase of modification of the TRADINNOVA algorithm in order to adapt it to the ACO metaheuristic. The preliminary studies augur good results of TRADINNOVA-ACO.

6 Conclusions and future research

In this paper it has been shown that dynamic management of stocks is necessary, because for this TRADINNOVA has been implemented. The results obtained by this algorithm are extremely encouraging and can be seen in [1].

This algorithm leaves the stock selection process up to the investor, who can use any technique. This has made it possible for us to introduce the ant colony optimization metaheuristic as criteria for selection. With this metaheuristic we achieve the optimum stock dealing path which an investor would take between specific dates, something which will help us in the selection process.

Future processes would mean the implementation of modifications in TRADINNOVA in order to adapt it to this metaheuristic and check whether the predictions made surpass the reference index. We also propose the use of other types of metaheuristics for this objective and, therefore, the carrying out of a comparative study of them when achieving the optimum stock dealing path.

Acknowledgements

The authors thank the "Ministerio de Educación y Ciencia" of Spain and the "European Fund for Regional Development" (FEDER) for the partial support given to this work under the project TIN2005-08404-C04-02.

References

1. I. J. Casanova and J. M. Cadenas. TRADINNOVA-Un algoritmo heurístico de compra-venta inteligente de acciones. *CM-ΠΙ 2006, Congreso Multidisciplinar de Percepción e Inteligencia*, Accepted and pending of publication, 2006.
2. M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, 1999. London, UK.
3. M. Dorigo, G. Di Caro and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5 (2): 137–172, 1999.
4. O. Cordón, F. Herrera and T. Sttzle. A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends. *Mathware and Soft Computing* 9(2-3):141–175, 2002.
5. E. P. K. Tsang and S. Martinez-Jaramillo. Computational Finance. Feature Article. *IEEE Computational Intelligence Society*, 2004.
6. K. Swingler. Financial Prediction, some pointers, pitfalls, and common errors. *Neural Computing and Applications*, 4(4):192–197, 1996.
7. M. Zekic. Neural Network Applications in Stock Market - A Methodology Analysis. *9th International Conference on Information and Intelligent Systems '98*, pages 255-263, 1998.
8. M. Januskeviciuts. Testing Stock Market Efficiency Using Neural Networks Case of Lithuania. *Thesis. SSE Riga Working Papers*, 17(52), 2003. http://www2.sseriga.edu.lv/library/working_papers/FT_2003_17.pdf

9. E. P. K. Tsang and J. Li. Combining Ordinal Financial Predictions with Genetic Programming. *IDEAL-2000, Second International Conference on Intelligent Data Engineering and Automated Learning*, pages 13–15, 2000.
10. A. E. Drake. Genetic Algorithms in Economics and Finance: Forecasting Stock Market Prices and Foreign Exchange - A Review. http://www.agsm.unsw.edu.au/~bobm/papers/drake.pdf
11. L. Davis. Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets. In G. J. Deboeck, editor, *PART II: 8. Genetic Algorithms and Financial Applications*. Wiley Finance Edition, 1994.
12. P. Kroha and R. Baeza-Yates. Classification of Stock Exchange News. *Technical Report. Department of Computer Science, Engineering School, Universidad de Chile*, 2004. http://www.tu-chemnitz.de/informatik/service/if-berichte/pdf/CSR-04-02.pdf
13. M. A. Mittermayer. Forecasting Intraday Stock Price Trends with Text Mining Techniques. hicss, p. 30064b. *HICSS'04, 37th Annual Hawaii International Conference on System Sciences - Track 3*, 2004.
14. C-C. Tseng. Comparing Artificial Intelligence Systems for Stock Portfolio Selection. *The 9th International Conference of Computing in Economics and Finance*, 2003.
15. G. Kendall and Y. Su. A Particle Swarm Optimization Approach in the Construction of Optimal Risky Portfolios. *The 2005 IASTED International Conference on Artificial Intelligence and Applications*, pages 40–45, 2005.
16. M. Rubinstein. Markowitz's "Portfolio Selection": A Fifty-Year Retrospective. *Journal of Finance*, 57(3):1041–1045, 2002.
17. A. Borodin, R. El-Yaniv and V. Gogan. Can We Learn to Beat the Best Stock. *Journal of Artificial Intelligence Research*, 21:579–594, 2004.

CHAC. A MOACO Algorithm for Computation of Bi-Criteria Military Unit Path in the Battlefield

***A.M. Mora*[1], *J.J. Merelo*[1], *C. Millan*[2], *J. Torrecillas*[2], *and J.L.J. Laredo*[1]**

[1] Departamento de Arquitectura y Tecnología de Computadores
University of Granada (Spain)
{amorag,jmerelo,juanlu}@geneura.ugr.es
[2] Mando de Adiestramiento y Doctrina
Spanish Army
{cmillanm, jtorrelo}@et.mde.es

Abstract. *In this paper we propose a Multi-Objective Ant Colony Optimization (MOACO) algorithm called CHAC, which has been designed to solve the problem of finding the path on a map (corresponding to a simulated battlefield) that minimizes resources while maximizing safety. CHAC uses a single colony and it has been implemented and tested with two different state transition rules: first one that combines the heuristic and pheromone information of both objectives and second one that is based on the dominance concept of multiobjective optimization problems. These rules have been evaluated in several different situations (maps with different degree of difficulty), and we have found that they yield better results than a greedy algorithm (taken as baseline) in addition to a* military *behaviour that is also better in the tactical sense. The rule which combines the information of both objectives, in general, yields better results than the one based on dominance.*

1 Introduction

Prior to any combat manoeuvre, the unit commander must plan the best path to get to a position that is advantageous over the enemy in the tactical sense. This decision is conditioned by two criteria, speed and safety, which he must evaluate. The choice of a safe path is made when the enemy forces situation is not known, so the unit must move through hidden zones in order to avoid detection, which may correspond to a very long, and thus slower, path. On the other hand, the choice of a fast path is made when the unit is going to attack the enemy or if there are few hidden (safe) zones in the terrain and going through it may produce a lot of casualties. In any case, the computation of the best itinerary for the unit to reach the objective point is a very important tactical decision. We will try to make this decision in the paper by solving what we have called the military unit pathfinding problem.

This problem is similar to the common pathfinding problem with some additional features and restrictions. It intends to find the best path from an origin to a destination point in the battlefield but keeping a balance between route speed and safety. The unit has an energy (health) and a resource level which are consumed when the unit moves through the path depending on the kind of terrain and difference of height, so the problem objectives are adapted to minimize the consumption of resources (which usually means walking as

short/fast as possible) and the consumption of energy. In addition, there might be some enemies in the map (battlefield) which could shoot their weapons against the unit.

To solve the problem an Ant Colony System (ACS) algorithm has been adapted. It is a type of ACO [1, 2], which offers more control over the exploration and exploitation parts of search. In addition, some features to deal with the two objectives of the problem have been added (in [3] it can be found a survey of MO algorithms), so it is a MOACO algorithm (paper [4] presents a review of some of them). This problem had been solved so far using classical techniques like branch and bound or dynamic programming, which do not usually scale well with size, but to the best of our knowledge find no one that treat the problem as a multiobjective and solve it with a MOACO.

2 The Problem

Our objective in this paper is to give the unit commander, or to a simulated unit in a combat training simulator, a tactical decision capability so that it will be able to calculate the best path to its target point in the battlefield considering the same factors that a commander would.

The battlefield has been modeled as a cell grid, every cell corresponding to a 500x500 meter zone in real world.

The speed in a path is associated with the unit resources consumption because we have assigned a penalization to every cell related to the 'difficulty' of going through it and we have called this penalization resource cost of the cell. So, going through cells with more resource cost is more difficult and so more slow. Due to this justification, we refers to fast paths or paths with small resource cost.

Thus, the problem unit has two properties, a number of *energy* (which represent global health of unit soldiers or status of the unit vehicles) points and a number of *resource* (which represent unit supplies such as fuel, food and even moral) points. Its objective is to get to a target point with the maximum level in both properties.

The unit energy level is decremented in every cell of the path (the company depletes its human resources or vehicles suffer damage), which have a penalization called *no combat casualties*, depending on its type. In addition there is an extra penalization due to the impact of an enemy weapon in the cell. This value is calculated as combination of three other, the probability of enemy shoots, the probability of impact in the unit, and the damage it would produce to the unit. Moreover every cell has an assigned resources penalty depending on the type of terrain. There is an extra penalty when the unit goes from a cell to other with different height (if it goes down, the penalty is small than if it goes up).

Besides *cost in energy*, and *cost in resources*, cells have the following properties:

- *Type*: there are four kinds of cells: normal (flat terrain), forest, water and obstacle, three different types of terrain and obstacle which means a cell that the unit cannot going through. There are several *subtypes*: it can be an enemy unit position, problem unit position, or the target point; it can also be affected by enemy fire (with one hundred levels of impact) or be lethal for the problem unit.
- *Height*: an integer number (between -3 and 3) which represents a level of depth or height of a cell.

We have implemented an application in Delphi 7 for Windows XP in order to create the problem scenarios (battlefields) and also to visualize the solutions obtained by CHAC. This

application is available under request. Figure 1 shows an example of battlefield. It includes all typical features: enemy unit, obstacles, forest and water filled terrain accidents.

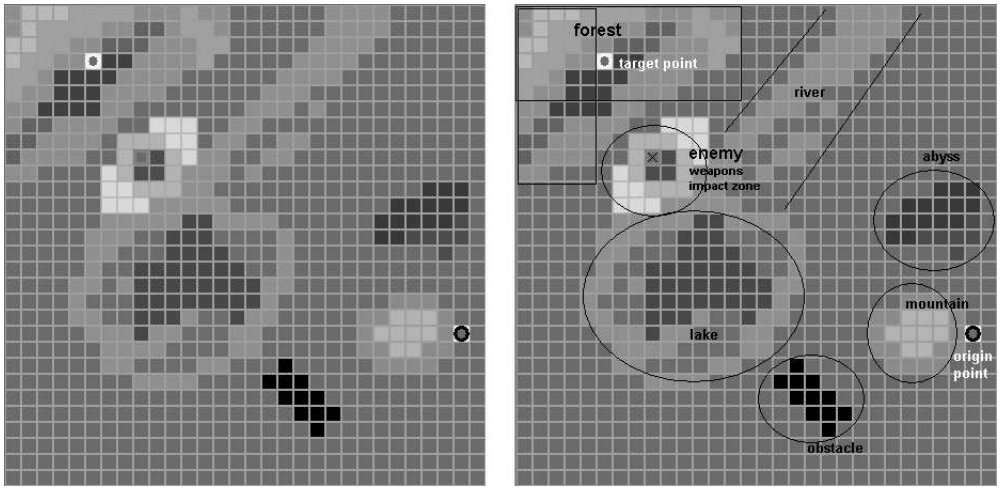


Fig. 1. Map Example. It is a 30x30 map which shows some of the types and subtypes above mentioned. In left, and explanation of cells types is showed, because it is difficult to differ between different kinds of cells due to the black and white image. The example includes the problem unit (circle with black border), an enemy unit (square with gray border and mark with 'X'), the zone affected by enemy weapons (in different shades of light gray depending on the associated damage) surrounding the enemy and the objective point for the unit (circle with light gray border). The different shades in the same color models height (light color) and depth (dark color).

This implementation includes some constraints: problem and enemy units fill up exactly one cell (which corresponds with their size in real world). The problem unit as well as the target point must be located on the map; the presence of the enemy is optional. The unit can go only once through a cell and cannot go through cells occupied by an enemy or an obstacle, or cells with a cost in resources or energy bigger than what the unit has available. Some rules also apply to the *line of sight*, so that it behaves as realistically as possible.

3 The CHAC Algorithm

CHAC means *Compañía de Hormigas Acorazadas* or Armoured Ant Company to relate the ACO algorithms with the military scope of this application. It is an ACS adapted to deal with several objectives, that is, a MOACO or Multi-objective Ant Colony Optimization algorithm.

In order to approach it via an ACO algorithm, the problem must be transformed into a graph with weighted edges. Every cell in the map is considered a node in the graph and 8 edges connect it to every one of its neighbours (except in border cells). To deal with two objectives, there are two weights in every edge related to the cost of energy and resources, that is, a cost related to the energy expenses (cost assigned to the destination node of the

edge) and other related to the consumption of resources if the unit moves from one cell to its neighbour following that edge (which depends on the types of both nodes and height difference between them).

The algorithm implemented within CHAC is constructive which means that in every iteration, every ant builds a complete solution, if possible (there are some constraints, for example the unit cannot go through one node twice in the same path), by travelling through the graph. In order to guide this movement, the algorithm uses information of two kinds: pheromone and heuristic, that will be combined.

Since it is a multiobjective problem, CHAC uses two pheromone matrices, one per objective, and two heuristics functions (also matrices), following the BicriterionAnt algorithm designed by Iredi et al. in [5]. We use a single colony too, however, we decided to use an Ant Colony System (ACS) instead of an Ant System to have better control in the balance between exploration and exploitation by using the parameter q_0. We have implemented two state transition rules (which means two different algorithms), first one similar to the Iredi's proposal and second one based on dominance of neighbours. The local and global updating formulas are based in the MACS-VRPTW algorithm proposed by Barán et al. in [6], with some changes due to the use of two pheromone matrices.

The objectives are named $\boldsymbol{f}$, minimization the resources consumed in the path (speed maximization) and $\boldsymbol{s}$, minimization the energy consumed in the path (safety).
The Heuristic Functions try to guide search between the start and the target point considering the most important factors for every objective. So, for edge *(i,j)* they are:

$$\eta_f(i,j) = \frac{\omega_{fr}}{Cr(i,j)} + \frac{\omega_{fd}}{Dist(j,T)} + (\omega_{fo} \cdot ZO(j)) \tag{1}$$

$$\eta_s(i,j) = \frac{\omega_{se}}{Ce(i,j)} + \frac{\omega_{sd}}{Dist(j,T)} + (\omega_{so} \cdot ZO(j)) \tag{2}$$

In Equation 1, *Cr* is the resource cost when moving from node i to node j, *Dist* is the Euclidean distance between two nodes (T is the target node of the problem) and *ZO* is a score (between 0 and 1) to a cell being 1 when the cell is hidden to all the enemies (or to all the cells in a radius when there are no enemies) and decreasing exponentially when it is seen. ω_{fr} , ω_{fd} and ω_{fo} are weights to assign relative importance to the terms in the formula. In this case, the most important term is the distance to target point because it searches for the fastest path. In second place is important the resources cost and only a little the hidden of the cell (it almost does not mind in a fast path).

In Equation 2, *Ce* is the energy cost of moving to node j, *Dist* and *ZO* is the same as the previous formula. ω_{se} , ω_{sd} and ω_{so} are again weights to assign relative importance to the terms in the formula, but in this case the most important are energy cost and hidden (both are to be considered in a safe path) and a little the distance to target point.

The *Combined State Transition Rule* (CSTR) is a formula used to decided which node j is the next in the construction of a solution (path) when the ant is at the node i, it is the pseudo-random-proportional rule used in ACS, but adapted to deal with a two objectives problem by combining the heuristic and pheromone information of both of them:

If (q $\leq$ q0)

$$j = \arg\max_{j \in N_i} \left\{ \tau_f(i,j)^{\alpha \cdot \lambda} \cdot \tau_s(i,j)^{\alpha \cdot (1-\lambda)} \cdot \eta_f(i,j)^{\beta \cdot \lambda} \cdot \eta_s(i,j)^{\beta \cdot (1-\lambda)} \right\} \tag{3}$$

Else

$$P(i,j) = \begin{cases} \dfrac{\tau_f(i,j)^{\alpha\cdot\lambda} \cdot \tau_s(i,j)^{\alpha\cdot(1-\lambda)} \cdot \eta_f(i,j)^{\beta\cdot\lambda} \cdot \eta_s(i,j)^{\beta\cdot(1-\lambda)}}{\sum_{u\in N_i} \tau_f(i,u)^{\alpha\cdot\lambda} \cdot \tau_s(i,u)^{\alpha\cdot(1-\lambda)} \cdot \eta_f(i,u)^{\beta\cdot\lambda} \cdot \eta_s(i,u)^{\beta\cdot(1-\lambda)}} & if\ j \in N_i \\ \\ 0 & otherwise \end{cases} \tag{4}$$

Where $q_0 \in [0,1]$ is the standard ACS parameter, q is a random value in [0,1]. τ_f and τ_s are the pheromone matrices and η_f and η_s are the heuristic functions for the objectives (Equations 1 and 2). All these matrices have a value for every edge *(i,j)*. α and β are the usual weighting parameters and N_i is the current feasible neighbourhood for the node i.
$\lambda \in (0,1)$ is a user-defined parameter which sets the importance of the objectives in the search (this is an application created for a military user who decides which objective has more priority), so for instance, if the user decides to search for the fastest path, λ will take a value close to 1; if he wants the safest path, it has to be close to 0. This value is kept during all the algorithm and for all the ants, unlike other bi-criterion implementations in which the parameter takes value 0 for first ant and it was growing for every ant until it takes a value 1 for the last one.

When an ant is building a solution path and it is placed at one node i, if $q \leq q_0$ the best neighbour j is selected as the next (Equation 3). Otherwise, the algorithm decides which node is the next by using a roulette considering *P(i,j)* as probability for every feasible neighbour j (Equation 4).

On the other hand, the *Dominance State Transition Rule* (DSTR) is based on the dominance concept (see reference [3]), which is defined as follows (a dominates b):

$$a \succ b \ \ if: \quad \forall i \in 1,2,...,k \mid C_i(a) \leq C_i(b) \quad \wedge \quad \exists j \in 1,2,...k \mid C_j(a) < C_j(b) \tag{5}$$

Where a and b are two different vectors of k values (one per objective) and C is a cost function for every component in the vector. If it intends to minimize the cost and Equation 5 is true, then b is dominated by a.

So, in our problem there are two cost functions to evaluate the dominance between nodes. Actually these functions consider edges because they have assigned pheromone and heuristic information, and combine them:

$$C_f(i,j) = \tau_f(i,j)^{\alpha\cdot\lambda} \cdot \eta_f(i,j)^{\beta\cdot\lambda} \tag{6}$$

$$C_s(i,j) = \tau_s(i,j)^{\alpha\cdot(1-\lambda)} \cdot \eta_s(i,j)^{\beta\cdot(1-\lambda)} \tag{7}$$

In addition there is a function which uses Equations 6 and 7:

$$D(i,j,u) = \begin{cases} 1 & if\ (i,j) \succ (i,u) \\ \\ 0 & otherwise \end{cases} \tag{8}$$

At last, the dominance state transition rule is as follows:

If (q $\leq$ q0)

$$j = \arg\max_{j \in N_i} \left\{ \sum_{u \in N_i} D(i,j,u) \quad \forall j \neq u \right\} \tag{9}$$

Else

$$P(i,j) = \begin{cases} \dfrac{\left(\sum_{u \in N_i} D(i,j,u) \right) + 1}{\sum_{k \in N_i} \left(\left(\sum_{u \in N_i} D(i,k,u) \right) + 1 \right)} & if\ j \in N_i \wedge j \neq u \wedge k \neq u \\ 0 & otherwise \end{cases} \tag{10}$$

Where all the parameters are the same as in Equations 3 and 4. This rule chooses the next node j in the path (when an ant is placed at node i) considering the number of neighbours dominated for every one. So if $q \leq q_0$, the node which dominates more of the other neighbours is chosen, otherwise the probability roulette wheel is used. In Equation 10 we add 1 to avoid a 0 probability if no one dominates other neighbour.

There are two *Evaluation Functions* (one per objective, again):

$$F_f(Psol) = \sum_{n \in Psol} [Cr(n-1,n) + \omega_{Ffo} \cdot (1 - ZO(n))] \tag{11}$$

$$F_s(Psol) = \sum_{n \in Psol} [Ce(n-1,n) + \omega_{Fso} \cdot (1 - ZO(n))] \tag{12}$$

Where $Psol$ is the solution path to evaluate and ω_{Ffo} and ω_{Fso} are weights related to the importance of visibility of the cells in the path. In Equation 11 its importance will be small, it is less important to hide in a fast path and it will be high in Equation 12 for the opposite reason. The other terms are the same that in Equations 1 and 2.

Since CHAC is an ACS, there are two levels of pheromone updating, local and global, which update two matrices at each level. The equations for *Local Pheromone Updating* (performed when a new node j is added to the path an ant is building) are:

$$\tau_f(i,j) = (1 - \rho) \cdot \tau_f(i,j) + \rho \cdot \tau_{0f} \tag{13}$$

$$\tau_s(i,j) = (1 - \rho) \cdot \tau_s(i,j) + \rho \cdot \tau_{0s} \tag{14}$$

Where ρ in [0,1] is the common evaporation factor and τ_{0f}, τ_{0s} are the initial amount of pheromone in every edge for every objective, respectively:

$$\tau_{0f} = \frac{1}{(numc \cdot MAX_R)} \tag{15}$$

$$\tau_{0s} = \frac{1}{(numc \cdot MAX_E)} \tag{16}$$

With *numc* as the number of cells in the map to solve, MAX_R as the maximum amount of resources going through a cell may require, and MAX_E as the maximum energy cost going through a cell may produce (in the worst case).

The equations for *Global Pheromone Updating* are:

$$\tau_f(i,j) = (1-\rho) \cdot \tau_f(i,j) + \rho/F_f \tag{17}$$

$$\tau_s(i,j) = (1-\rho) \cdot \tau_s(i,j) + \rho/F_s \tag{18}$$

Only the solutions inside the Pareto set will make the global pheromone updating once all the ants have finished of building paths in every iteration.

The *Pseudocode* for CHAC is as follows:

```
Initialization of pheromone matrices with initial amount   /* Equations 15,16 */
For i=1 to NUM_iterations
   For a=1 to NUM_ants
      ps=build_path(a)   /* Equations 1,2, [(3,4) or (6,7,8,9,10)] , 13,14 */
      evaluate(ps)   /* Equations 11,12 */
      if ps is non-dominated
         insert ps in Pareto_Set
         remove from Pareto_Set dominated solutions
      endif
   EndFor
   global_pheromone_updating   /* Equations 17,18 */
EndFor
```

4 Experiments and Results

We would like to first emphasize that the values for the parameters and weights affect the results because parameters guide the search and balance the exploration and exploitation of the algorithm, and weights set the importance of every term in the heuristics and evaluation functions. We fine-tuned both in order to obtain a good behaviour in almost all the maps; the values found were $\alpha = 1$, $\beta = 2$, $\rho = 0.1$ and $q_0 = 0.4$, in the three experiments we show in this section. The last value establishes a balance between exploration and exploitation, tending to exploration (but without abandoning exploitation altogether). The weights described in Equations 1, 2, 11 and 12 are set to give more importance to minimizing distance to target point and consumption of resources in the speed objective, and to give more importance to minimizing visibility and consumption of energy in the safety objective.

The user can only decide the value for λ parameter, which gives relative importance to one objective over the other, so if it is near 1, finding fastest path would be more important and if it is near 0, the other way round.

As every MO algorithm, CHAC yields a set of non-dominated solutions from which the user chooses using his own criteria, since, usually he only wants one solution. But in this algorithm the resulting Pareto set is small (about five to ten different solutions on average, depending on the map size) because it only searches in the region of the ideal Pareto front determined by the λ parameter. We are going to represent in the results only 'the best'

solution inside all the Pareto sets of 30 executions, considering better the solution with the smallest cost in the most significant criteria (depending on λ value). In order to clarify this concept, there is not a best path, but paths enough good from the military point of view. These paths have been selected by the military participation of the project taking into account tactical considerations and the features of every battlefield.

CHAC have been implemented using the two state transition rules as two different algorithms: CSTR which uses the *combined state transition rule* and DSTR which uses the *dominance state transition rule*. We are going to compare the results between both CHAC implementations (with the same parameter values), and both of them with a greedy approach which uses the same heuristic functions as cost functions (the *pheromones* has no equivalent in this algorithm), but using a *dominance criterion* to select the next node of the actual cell neighbourhood (it selects as next the node which dominates most of the others neighbours considering their cost in both objectives). This algorithm is quite simple, and sometimes does not even reach a solution because it gets into a loop situation (back and forth between a set of cells consuming all the resources or energy).

We have performed experiments on three different maps; two of them have cells of a single type of terrain (with different heights) in order to avoid problems of visualization due to the black and white figures. We executed every CHAC implementation 30 times with each extreme value for λ parameter (0.9 and 0.1) in order to search for the fastest path and the safest path (they cannot be 1 and 0 because always the two objectives must be considered). In every execution we chose the best solution (enough good from a military point of view) in the Pareto set considering the objective we are minimizing and once we have finish all the 30 executions, we made mean and standard deviation of them.

4.1 River and Forest Map

The first scenario is a 15x15 map with a river flowing between the start and the target point and a patch of forest. There is no enemy. The best of fast paths found and the best of safe paths found are shown in Figures 2 and 3, one for each implementation, marked by circles.

	Combined State Transition Rule				*Dominance State Transition Rule*				*Greedy*
	Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)		Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)		
	F_f	F_s	F_f	F_s	F_f	F_s	F_f	F_s	
Best	**22.198**	85.768	34.899	**61.189**	**22.214**	86.077	37.917	**61.833**	
Mean	**22.207**	85.941	28.505	**67.296**	**22.595**	88.405	28.724	**66.365**	NO SOLUTION
	±0.022	±0.436	±1.993	**±3.360**	**±0.555**	±3.127	±2.637	**±2.881**	

Table 1. Results for River and Forest map. (500 iterations, 20 ants)

As it can be seen in Table 1, the cost of the solutions for the fastest path have a small standard deviation which means possibly CHAC has reach a quasi-optimal solution (small search space and enough iterations and ants to solve it) in both implementations of state transition rule (greedy does not reach a feasible solution). Mean and standard deviation in the objective which is not being minimized are logically worse, because it has little importance. DSTR even improving sometimes mean and standard deviation values in main objectives so, its solutions are more similar between executions (the algorithm has robustness).

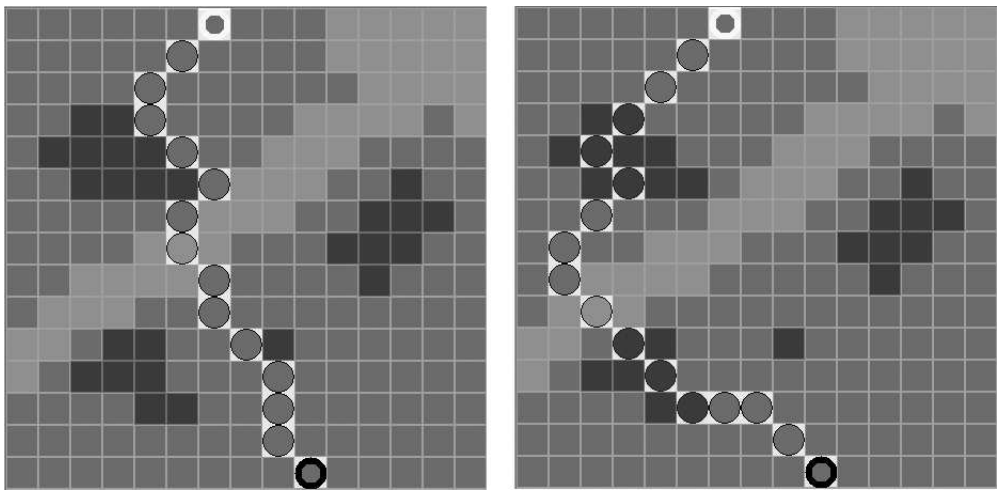


Fig. 2. [CSTR] River and Forest map (forest cells in dark gray and water cells in light gray). No enemy. Both use forest cells to avoid being seen, but the latter goes through them to optimize safety. Fastest (left) and safest (right) paths found.

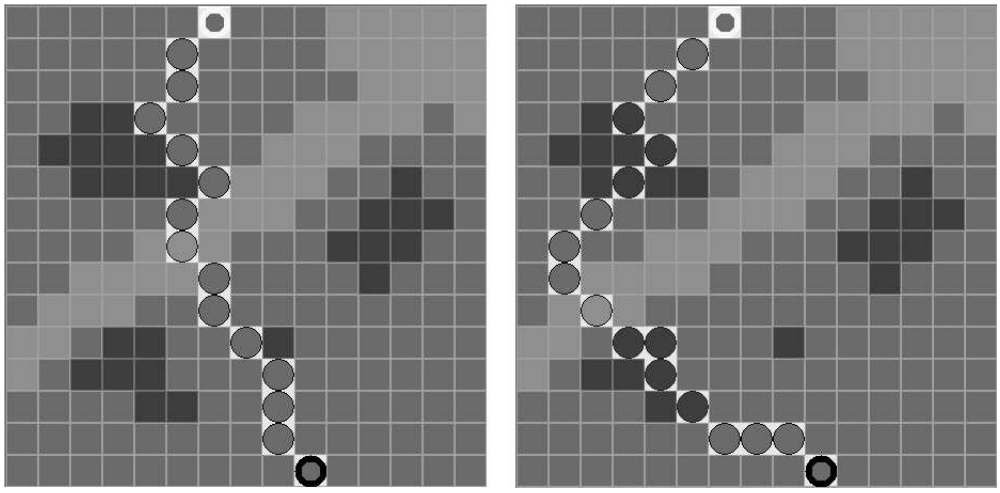


Fig. 3. [DSTR] River and Forest map (forest cells in dark gray and water cells in light gray). No enemy. Similar behaviour as in previous case, using forest to hide. Fastest (left) and safest (right) paths found.

Figure 2 (left) shows the fastest path found with CSTR, which goes zigzagging, because the algorithm usually moves to the most hidden cells when there are no enemies (in a radius that can be set up by the user and which is 10 in this experiment), even in the search for the fastest path, but with less relevance. Forest cells obstruct the line of sight so the unit tends to move near them while goes rather directly to the target point. Figure 2 (right) shows the safest path found which moves in a less straightforward way and hides by moving

near, and even inside forest cells (hidden is very important in safe paths) although it means bigger resource cost (forest cells have more resource cost assigned than flat terrain). Results showed in Figure 3 (left and right), corresponding to DSTR algorithm are similar, because the space problem is quite small.

4.2 Flat Terrain with Walls Map.

The second map contains 30x30 cells, and represents a flat terrain with some 'walls', there is one enemy unit protecting the target cell (there are some cells affected by its weapons fire). The best of fast paths found and the best of safe paths found are shown in Figures 4 and 5, one for each implementation. Cells with light gray border in the path are hidden to enemy.

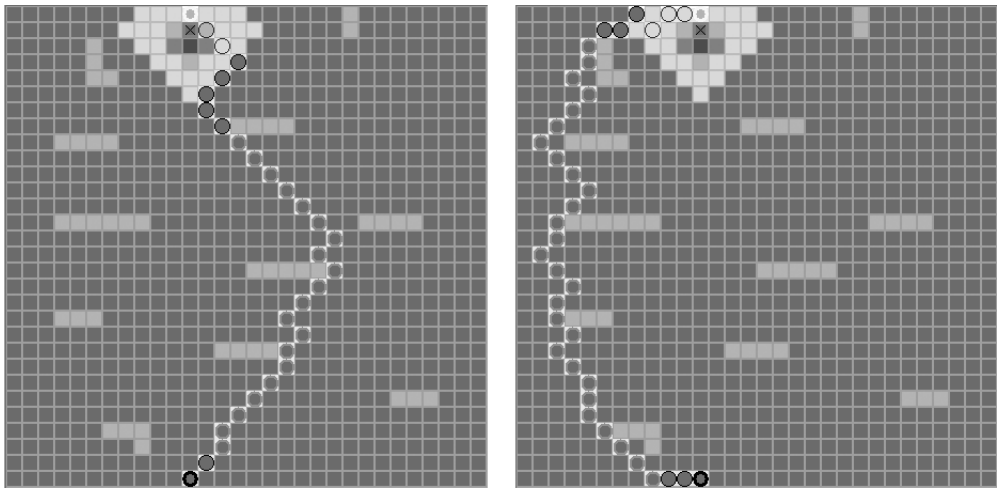


Fig. 4. [CSTR] Flat Terrain with Walls, and one enemy shooting in a zone near the target point (the enemy is marked with 'X', the cells surrounding it in some shades of gray are the zone of weapons impact and the other cells in light gray are the walls). Fastest (left) and safest (right) paths found by CHAC.

	Combined State Transition Rule				*Dominance State Transition Rule*				*Greedy*	
	Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)		Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)			
	F_f	F_s	F_f	F_s	F_f	F_s	F_f	F_s	F_f	F_s
Best	**36.500**	142.900	44.500	**112.700**	**39.000**	133.100	51.500	**113.400**		
Mean	**37.220**	141.973	45.650	**121.210**	**43.350**	240.660	59.530	**123.980**	46.7	322.9
	±1.216	±2.904	±3.681	**±10.425**	**±1.592**	±85.254	±10.174	**±8.119**		

Table 2. Results for the Flat Terrain with Walls map. (1000 iterations, 30 ants)

As it can be seen in Table 2, CHAC (in the two implementations) outperforms the greedy algorithm by almost 25% in resource cost and 75%. It is because the greedy path

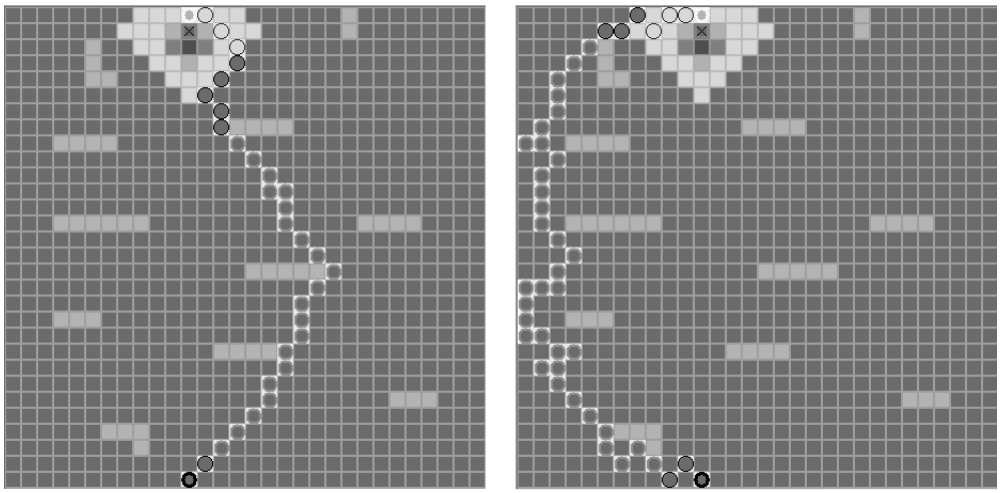


Fig. 5. [DSTR] Flat Terrain with Walls, and one enemy shooting in a zone near the target point (the enemy is marked with 'X', the cells surrounding it in some shades of gray are the zone of weapons impact and the other cells in light gray are the walls). Fastest (left) and safest (right) paths found by CHAC.

is rather straight and it means low resource expenses, but at the same time, it means the enemy sees the unit all the time (it moves through uncovered cells) and the safety cost is dramatically increased (it depends too much on cell visibility). The cost for fast path, which also depends on visibility, is increased too. The standard deviation is small in CSTR, but it grows a little in values of safest paths which could mean a bigger exploitation factor is needed. Again, in the DSTR implementation, results are similar to those obtained with CSTR, but with better standard deviation; however, its performance is poorer (even quite bad in one case) for the objectives it is not minimizing, which supports the theory that a higher exploitation level is needed in this scenario. In addition if we look at the fastest solution for DSTR we can see that it dominates the safest one (is better even in F_s cost), but it is an exception (if look at the mean value we can note this). It occurs because this method has a greater exploration component and it can be reach a different local optima in other search space area.

In Figure 4 (left), we can see the fastest path found with CSTR; the unit moves in a curve mode in order to hide from enemy behind the walls (it consider the hidden of cells from the enemy cell). It surrounds this terrain elevations because resource cost of going through them is big. On the other hand, in Figure 4 (right) we can see the safest path found with CSTR which moves surrounding the left side walls but remaining during more cells hide for the enemy. In both cases, the unit moves inside the zone affected by weapons choosing the less damaging cells, even in the safe case it arrives by a flank of the enemy unit where they have less fire power and this inflict less damage. It represents a good behaviour, very tactical in the case of safest path because attack to an enemy by the flank is one principle of military tactics. The results for DSTR is showed in Figure 5 (left and right) are similar but a little worse because the paths has some extra cells due to the need of a greater exploitation component in this implementation.

4.3 Valleys and Mountains Map.

This 45x45 map represents some big terrain accidents, with clear zones representing peaks (the more higher the more clearer) and dark zones representing valleys or cliffs. There are two enemies located in both peaks and the target point is behind one of them. The best of fast paths found and the best of safe paths found are showed in Figures 6 and 7, one for each implementation. Circle cells mark the paths, those with light gray border in the path are hidden to both enemies.

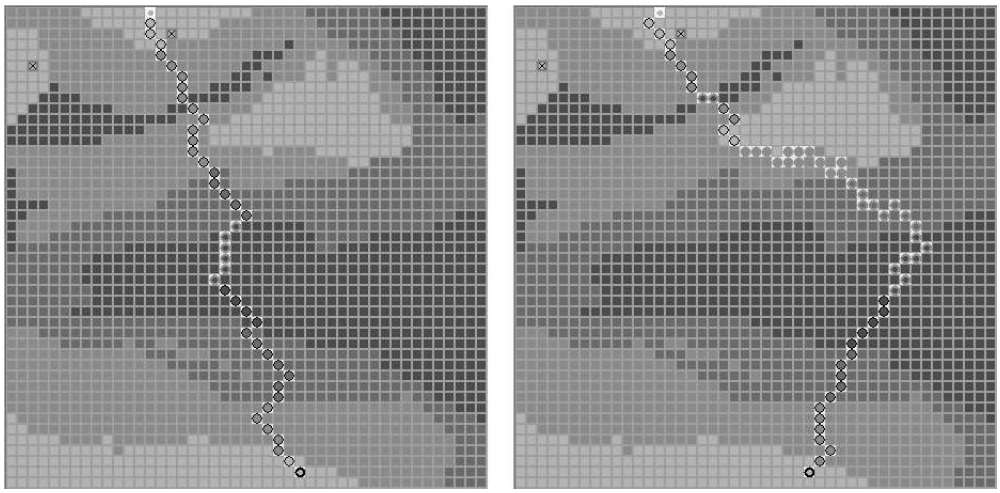


Fig. 6. [CSTR] Valleys and Mountains, with 2 enemy units (mark with 'X') on watch. Fastest (left) and safest (right) path found.

	Combined State Transition Rule				*Dominance State Transition Rule*				*Greedy*
	Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)		Fastest ($\lambda = 0.9$)		Safest ($\lambda = 0.1$)		
	F_f	F_s	F_f	F_s	F_f	F_s	F_f	F_s	
Best	**70.500**	374.300	88.500	**285.800**	**73.500**	394.500	77.000	**354.600**	
Mean	**75.133**	357.800	105.517	**311.390**	**77.950**	397.280	88.780	**371.890**	NO SOLUTION
	±2.206	±9.726	±17.138	**±19.541**	**±1.724**	±11.591	±13.865	**±7.522**	

Table 3. Results for Valleys and Mountains map. (2000 iterations, 70 ants)

As Table 3 shows, both implementations yield a low standard deviation, which means the algorithm is robust with respecto to initialization. CSTR implementation results may be nearer an optimal path, but in the DSTR case, the solutions have big costs (compared with the others), which means it needs more iterations or a greater exploitation factor to improve them. As in previous experiments, mean and standard deviation in the objectives that the algorithm is not minimizing are worse, because they have little importance. In this case, the differences between the cost of an objective when search minimizing this objective

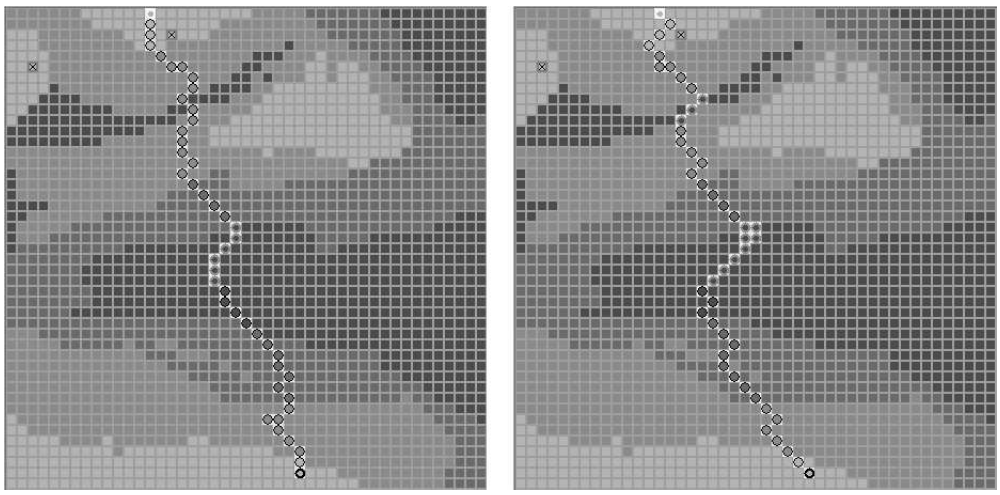


Fig. 7. [DSTR] Valleys and Mountains, with 2 enemy units (mark with 'X') on watch. Fastest (left) and safest (right) path found.

and when search minimizing the other are bigger than in the previous maps because the search space is bigger too.

In Figure 6 (left) we can see the fastest path found with CSTR, the unit goes in a rather straight way to the target point, but without considering the hidden of cells (from the position of both enemies) so the safety cost increases dramatically. On the other hand, in Figure 6 (right) we can see the safest path found with CSTR which represents a curve (distance to target point has little importance) which increases speed cost, but the unit goes through many hidden cells. This behaviour is excellent from military tactical point of view. Figure 7 shows the best solutions for DSTR and in this case we can see that fastest path (left) is similar to the fastest of CSTR, but safest path (right) is worse; it is too straight which means the enemies see the unit more time and it increases the safety cost. It points to DSTR does not work properly in big search spaces.

5 Conclusions and Future Work

In this paper we have described a MOACO algorithm called CHAC which tries to find the fastest and safest path, whose relative importance is set by the user, for a simulated military unit. This algorithm can use different state transition rules, and, in this papeer, two of them have been presented and tested. The first one combines heuristic and pheromone information of two objectives (Combined State Transition Rule, CSTR) and the second one is based on dominance over neighbours (Dominance State Transition Rule, DSTR).

The algorithm using both state transition rules has been tested in several different scenarios yielding very good results in a subjective assessment by the military staff of the project (Mr. Millán and Torrecillas) and being perfectly compatible with military tactics. They even offer good solutions in complicated maps in less time than a human expert would need. Moreover it is possible to observe an inherent emergent behaviour studying the solutions because it tends to be similar to those a real commander would, in many cases, take.

In addition CHAC (using any state transition rule of the ones tested so far) outperforms a greedy algorithm. In the comparison between them, CHAC with CSTR yields better results in the same conditions, but CHAC with DSTR is more robust, yielding solutions that perform similarly, independently of the random initial conditions. If we increase exploitation level or iterations of algorithm, DSTR approach offers similar results to CSTR.

As future work, we will compare CHAC with pathfinding algorithms better than the simple greedy we have used here. From the algorithmic point of view, we will try to approach more systematically parameter setting, and investigate its performance in dynamic environments where, for instance, the enemy can move and shoot on sight. We will also try to evaluate its performance in environments with a hard time constraint, that is, in environments where the algorithm cannot run for an unlimited amount of time, but a limited and small one, which is usually the case in real combat situations. From the implementation point of view, it would be interesting to implement it within a real combat training simulator, that includes realistic values for most variables in this algorithm, including fuel consumption and casualties caused by projectile impact.

We will also try to approach scenario design more systematically, trying to describe its difficulty by looking at different aspects. This will allow us to assess different aspects of the algorithm and relate them to scenario difficulty, finding out which parameter combination is better for each scenario.

Acknowledgements

This work has been developed within the SIMAUTAVA Project, which is supported by Universidad de Granada and MADOC-JCISAT of Ejército de Tierra de España. Mr. Millán is a Lieutenant Colonel and Mr. Torrecillas is a Major of the Spanish Army Infantry Corps.

References

1. M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.
2. M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In G.A. Kochenberger F. Glover, editor, *Handbook of Metaheuristics*, pages 251–285. Kluwer, 2002.
3. Carlos A. Coello Coello, David A. Van Veldhuizen, and Gary B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, 2002.
4. C. García-Martínez, O. Cordón, and F.Herrera. An empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. In *ANTS 2004. Fourth International Workshop on. Ant Colony Optimization and Swarm Intelligence*, number 3172 in LNCS, pages 61–72. Springer, 2004.
5. Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne, editors, *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, volume 1993 of *Lecture Notes in Computer Science*, pages 359–372, Berlin, 2001. Springer-Verlag.
6. B. Barán and M. Schaerer. A multiobjective ant colony system for vehicle routing problem with time windows. In *IASTED International Multi-Conference on Applied Informatics*, number 21 in IASTED IMCAI, pages 97–102, 2003. `http://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-1442302509&pa%rtner=40&rel=R4.5.0`.

A Simulation-based Ant Colony Algorithm for Assembly Line Buffer Allocation

Ivan Vitanov and John Kay

School of Applied Sciences
Cranfield University

Abstract. *This paper presents an algorithm for the near-optimal allocation of buffer space to an assembly line by means of the ant colony optimisation (ACO) paradigm. Uniquely, the algorithm has been designed to work in conjunction with a simulation model and is adapted to have both combinatorial and stochastic problem-solving capability. The simulation model was developed using the WITNESS simulation package and serves the purpose of an objective function evaluator, encapsulating the dynamics of the line and enabling the production rate for a particular buffer configuration to be determined. Finally, the WITNESS Optimiser module was used as a benchmark in validating the ACO algorithm's performance. In the simulation experiments conducted, ACO attained slightly better results overall.*

1 Introduction

The buffer allocation problem is a difficult optimisation problem in manufacturing systems design for which only approximate solutions are currently available. It is thus considered an 'open' problem and remains the subject of considerable research activity.

In analysing production systems there are two broad categories of approaches. Firstly, there are the analytical approaches, which allow a closed-form expression of the solution to be derived. These are somewhat limited in scope as they tend to break down on larger problem instances. Secondly, there are numerical approaches which make use of recursive computations.

When dealing with a complex system governed by random disturbances or events, discrete-event simulation is often the numerical tool of choice in devising a coherent system representation. Oftentimes, simulation permits the analysis of systems that are too difficult to model by other means. In designing a new system, the best combination of design variables (or inputs) is investigated to optimise performance measures (or outputs) of the simulation model. This procedure is known as simulation optimisation. Within it, the simulation model is viewed as a stochastic objective function and is integrated with an optimisation procedure in a 'closed-loop' arrangement, using feedback from the simulator to inform the search strategy of the optimiser. The symbiotic relationship between the simulation and optimisation models is illustrated in figure 1.

Recently, so-called metaheuristic approaches such as genetic algorithms, simulated annealing, tabu search and others, have been successfully applied to solving simulation optimisation problems. One such approach of more recent date, which is presently undergoing transition from purely deterministic combinatorial optimisation to its stochastic and simulation-based counterparts, is ant colony optimisation (ACO). This approach takes inspiration from the foraging behaviour of colonies of ants and applies the same metaphor

of distributed cooperative learning by a number of independent agents to the task of optimising discrete-parameter problems. Recent studies have tested this approach on a number of paradigmatic combinatorial problems such as the travelling salesman problem, quadratic assignment problem and others, revealing better-than-average or competitive results as compared with other metaheuristics.

In the present study, an ant colony algorithm has been developed to optimise the buffer sizes in an asynchronous assembly line of closed-loop type comprising ten machines and as many buffers. The line itself was modelled using the WITNESS simulation software. The algorithm was eventually tested against WITNESS's own thermostatistical simulated annealing optimisation routine.

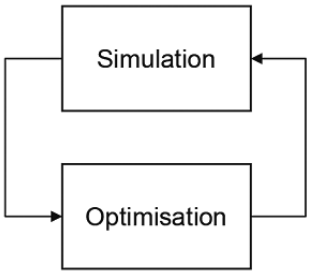


Fig. 1. A simulation optimisation model

2 Problem Description

An assembly line is defined as an "integrated structure of machines and operators that achieves construction of subsystems or finished products with specific characteristics using components or formless materials" (Bulgak et al, 1995). Assembly lines can be built according to various topologies. They may be constructed to include feedback or feed-forward loops, with a serial structure or with some machines in parallel. Part transfer is generally implemented using transfer chain, conveyor-type mechanisms or automated guided vehicles. Assemblies are frequently transported on work carriers or pallets with fixtures and/or jigs to hold component parts. In closed-loop asynchronous assembly systems, a fixed number of pallets or carriers are circulated ad infinitum by the material handling system. By reason of their asynchronicity, such lines allow for the independent movement of pallets and are therefore unreliable due to random fluctuations in processing times (as well as machine failures).

Assembly stations can be made less interdependent through the introduction of buffers, i.e. storage spaces for in-process inventory, whose storage and replenishment functions offset the propagation of machine idle-time. The buffer allocation problem consists in finding an optimal configuration of buffer sizes so as to maximise a certain performance measure such as the steady-state production rate of the line.

In the assembly line considered in this study (shown in figure 2 below), a set of assembly stations are arranged in series in the order of assembly operations performed. The stations are connected by conveyors.

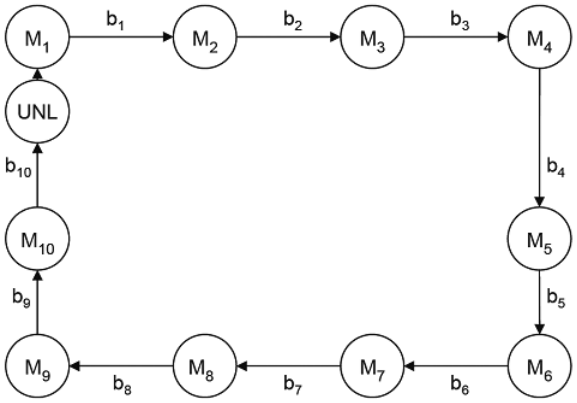


Fig. 2. Schematic of the assembly line

As can be seen from the figure, the modelled line is a single-loop system having ten machines and ten intermediate buffers, labelled M_1, M_2,..., $M_{10}$and b_1, b_2,..., b_{10} respectively. There is also an unload station before the first machine. A fixed number of pallets (this number can be varied in the simulation model) circulate perpetually in the line. Assemblies are carried around by the pallets, following a routing that takes them from the first through to the last station in sequence, with the finished assembly being removed at the unload station, leaving an empty pallet to return to the first machine where the assembly process starts over. Clearly, the number of pallets affects the production rate of the line.

The pallet dimensions and the span of the conveyor between any two adjacent stations determine the number of pallets that can be accommodated between these two adjacent stations. The maximum quantities for these work-in-process (WIP) inventories between each pair of stations represent the buffer sizes (capacities).

The task of buffer allocation in conveyor-driven systems such as the one investigated is complicated by the need to account for random fluctuations due to transport delays, in addition to fluctuations in production and failure rates. Transportation delays are proportional to buffer size, since a longer conveyor revolves over a greater distance and the speed of revolution is constant across all conveyors. The danger of having buffer sizes that are too large is that transport delays will be excessive and more in-process inventories must be input into the system to mitigate the large buffer sizes. When the buffer sizes are too small, then small processing delays will cause the buffer to fill up. With a fixed number of pallets in the system, there is always an optimal buffer configuration able to limit blocking and starvation effects considerably so as to generate a maximum possible steady-state production rate.

The simulation model is characterised by the following parameters:

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
Unbalanced (secs)	110	110	120	90	100	90	80	80	120	90
Balanced (secs)	100	100	100	100	100	100	100	100	100	100

- *Processing* or *cycle times* This is the time required for a station to perform an assembly task. Processing times are taken to be normally distributed random variables sampled according to a truncated normal distribution, with a variance of 10 seconds for all machines and ± 20 seconds upper and lower limit. Two separate line configurations (balanced and unbalanced) were tested, distinguished by different mean processing times as laid out in the table below. The values for the unbalanced line were randomly generated within realistic bounds.

- *Jam rate* Stations are subject to jamming due to various reasons such as system breakdowns, defective parts being assembled or a robotic assembly station accidentally dropping the part it is holding. Jam occurrences are random events. They are modelled as an average breakdown interval for each machine, calculated in terms of parts produced. The negative exponential distribution was used to describe the jam rate. The individual jam times were again randomly generated within realistic limits.
- *Jam clear time* This is the average time it takes to clear up a jam at a workstation expressed in seconds. The jam clear times are random variables modelled on the Erlang distribution. They were similarly generated randomly between realistic limits.
- *Delayed buffer* The buffers are intended to imitate the actions of conveyors and have a minimum delay time which is proportional to the buffer space: each additional unit of buffer space adds 7 seconds to the delay time. The buffer sizes between pairs of stations are the decision variables and can vary between 1 and 10 (with a step size of 1) for each buffer. For a line with 10 buffers this means there are 10^{10} possible buffer configurations.

The expected value of the production rate is represented by the number of parts produced in 100 000 simulation seconds (the duration of a simulation run), after a warm-up period of 10 000 seconds to remove the initial transient. At the end of each run the model is rewound to time zero in preparation for the next run. Independence of runs is maintained through incrementing the random number stream and substream for each random variable in the model.

3 Literature Review

Pre-90s research into production lines focused on predicting (evaluating) the performance characteristics of a given system configuration, rather than on the more practical goal of (optimising) system design, i.e. efficiently allocating buffer space. Since then a number of papers on the optimisation of such systems have appeared. Extensive literature reviews can be found in Gershwin (1994), Papadopoulos and Heavey (1996) and Altiok (1996).

In the context of simulation optimisation, various approaches have been examined in the literature. These include standard non-linear programming techniques, response surface methodology, stochastic approximation techniques and random search. A criticism levelled at traditional approaches such as these is that they tend to converge on local optima. Additionally, the likes of stochastic approximation and the response surface method are not best suited for application to combinatorial optimisation problems, including the buffer allocation problem. Moreover, commercial simulation optimisation software has been dominated by metaheuristic approaches, which though lacking in theoretical convergence guarantees have nevertheless proven robust in overcoming local optima.

Comprehensive reviews of the literature on simulation optimisation can be found in Swisher et al (2000) and Fu (2002). A systematic survey article on metaheuristics is provided by Blum and Roli (2003). Results on the application of genetic algorithms and simulated annealing to the buffer allocation problem are presented in Spinellis and Papadopoulos (1999 and 2000) and Bulgak et al (1995). A thorough overview of the ant colony metaheuristic can be found in Dorigo and Di Caro (1999).

4 Algorithmic Methodology

The software implementation of the optimisation algorithm and its synchronisation with the simulation model involved the use of a number of software packages besides WITNESS; namely, Matlab, Visual Basic and Microsoft Excel.

The software application consists of the assembly line model developed in the WITNESS simulation environment, the optimisation program - written mostly in Matlab code and executed from a Visual Basic module, and an Excel front end to display the results of the optimisation and simulation routines. The data exchange interfaces between the separate components of the application are displayed in figure 3 overleaf. The optimisation algorithm had to be developed independently of the WITNESS environment because the capabilities of WITNESS do not extend to general-purpose numerical programming. Instead, Matlab, a high-performance language for technical computing, was chosen as a suitable implementation platform.

The program code and scheme for the algorithm are given in Vitanov (2006). The algorithm draws on the work of Gutjahr and the S-ACO algorithm developed by him (Gutjahr, 2004) which represents the first general-purpose ant algorithm that tackles stochastic combinatorial problems, i.e. combinatorial optimisation under uncertainty.

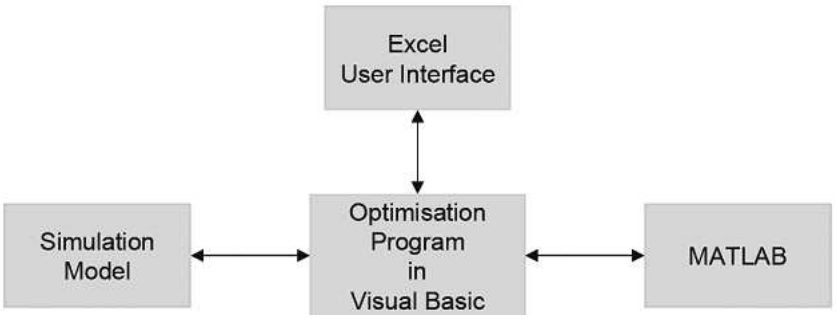


Fig. 3. Top-level view of the application

The buffer algorithm performs a search in the feasible solution space looking for an optimal solution. In this context, a solution can be any configuration of buffer sizes (b_1, b_2,..., b_{10}), as long as the values they take do not exceed the limits set on individual buffer capacity, which, as previously noted, could range from 1 to 10 pallet spaces.

The stepwise construction of a solution is represented by a random walk in the construction graph C, beginning in the start node. Following the definition of the construction graph encoding given in figure 4, the walk must satisfy the condition that each node is visited at

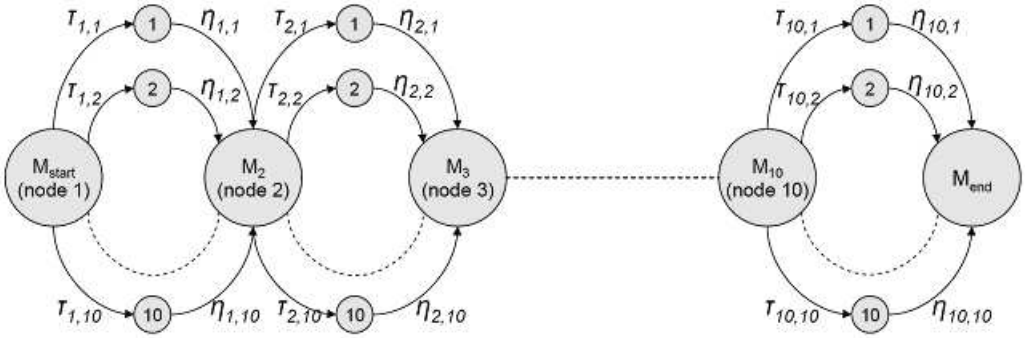


Fig. 4. Construction graph for the buffer allocation problem

most once; already visited nodes are infeasible. When there is no feasible unvisited successor node available, the walk stops and is decoded as a complete solution for the problem. The conceptual unit performing the walk is termed an artificial ant.

To each feasible walk in the sense above, there corresponds exactly one feasible solution. When constructing a walk in the construction graph, the probability $p_{i,j}$ to go from a node i to a feasible successor node j is chosen as proportional to $[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}$ where $\tau_{i,j}$ is the pheromone value and $\eta_{i,j}$ is the heuristic value. The complete formula is:

$$p_{i,j} = \begin{cases} 0, \; if(i,j)\, is\, not\, feasible, \\ \frac{[\tau_{i,j}]^{\alpha}[\eta_{i,j}]^{\beta}}{\sum_{(i,k)}[\tau_{i,k}]^{\alpha}[\eta_{i,k}]^{\beta}} \end{cases}$$

As in some other frequently used ACO variants for deterministic problems, in each round a round-winner is determined. In the stochastic context, this is done by comparing all walks that have been performed in this round on a single random scenario, drawn specifically for this round. In an ACO implementation for a deterministic problem, it is always reasonable to store the best solution seen so far in a special variable. A crucial difference to the deterministic case is that in a stochastic context it is not possible anymore to decide with certainty whether a current solution is better than the solution presently considered as the best found. Only a 'good guess' can be made by sampling. After a current round-winner has been determined, it is compared with the solution considered currently as the overall best solution. This is done based on a sample of $\omega = 5$ randomly drawn scenarios used by both solutions. Also, these scenarios are round-specific, i.e. in the next round new scenarios will be drawn. The winner of the comparison is stored as the new 'global best' solution. The round winner and global best are the only solutions whose constituent arcs are updated with pheromone.

The optimisation algorithm itself has certain parameters which need to be adjusted in order to get the best performance out of it. These are: alpha and beta, the pheromone and heuristic weights respectively; rho, the pheromone evaporation rate; and c1 and c2, the global-best and local-best reinforcement. These parameters were tuned using experimental designs and the determination of a response surface metamodel.

5 Experimental Results

The line model considered in this text has a solution space comprising 10^{10} solutions. Because of its magnitude, it was not possible to know in advance what an optimal solution might look like. To validate the ACO algorithm's performance therefore and to be sure of finding a close-to-optimal design for the system under study, it was decided to pit the algorithm against an established simulation-based metaheuristic technique with known performance guarantees. This served the double purpose of determining the fitness of the ACO algorithm in solving the buffer allocation problem *and* gauging its suitability for simulation optimisation. As already stated, these issues were addressed by running tests with both the WITNESS Optimiser module and the ACO algorithm on the assembly line model developed in WITNESS. To this end, the WITNESS Optimiser and ACO algorithms were compared over a set of 10 simulation experiments each, for both the balanced and unbalanced line configurations. In each case, five different pallet inputs were considered. To ensure a fair comparison, both algorithms were allowed to complete exactly 1000 solution evaluations in each experimental instance.

The runtime of the algorithms could not be defined in terms of real time or number of iterations, because one or the other algorithm would be disadvantaged. Since ACO uses a population of solutions and each ant constructs an individual solution, then the number of ants defines the number of evaluations per iteration. On the other hand, WITNESS Optimiser performs only one evaluation per iteration. With respect to actual running time, the WITNESS Optimiser is at a distinct advantage because it is integrated into the WITNESS environment itself while the ACO algorithm is spread across four different applications, so running times are not comparable. That being said, overall running times were not exorbitantly different between the two algorithms: WITNESS Optimiser took approximately 1hour 30minutes to complete 1000 evaluations, whereas ACO took around 1hour 45 minutes.

In each experimental instance, after the algorithms had completed the set number of evaluations and had yielded their respective best solutions, i.e. best buffer configurations found, they were then compared on the basis of 10 replications executed with the simulation model, from which an average response value was calculated for each solution. The replications were performed using the same set of random number scenarios for both solutions being compared.

For the simulation model used, thirteen was found to be a critical number of pallets in relation to system performance. Including fewer pallets meant getting a trivial result, with the smallest buffer configuration, i.e. all buffer sizes equal to one, being found optimal. This occurred, since queueing effects were rendered almost non-existent because of too few pallets, so that transportation times, which are contingent on buffer size, had an overwhelming impact on production rate. With pallets > 13, the randomness and general noise in the system seemed to increase cumulatively with each additional pallet, so that thirteen could be seen almost as a tipping point in terms of stability

The results of the simulation experiments are shown in figure 5. For the unbalanced line, WITNESS Optimiser outperformed ACO in three of the five test instances, though its performance was bettered on the remaining two. Overall, the results were closely matched. The best system design of those tested would appear to be in the case of 33 pallets.

In the next batch of experiments, those for the balanced line, ACO came out on top in four of the five trials. That means that over the 10 test instances investigated, ACO leads WITNESS Optimiser 6:4. Of course, more tests would have to be conducted for the results

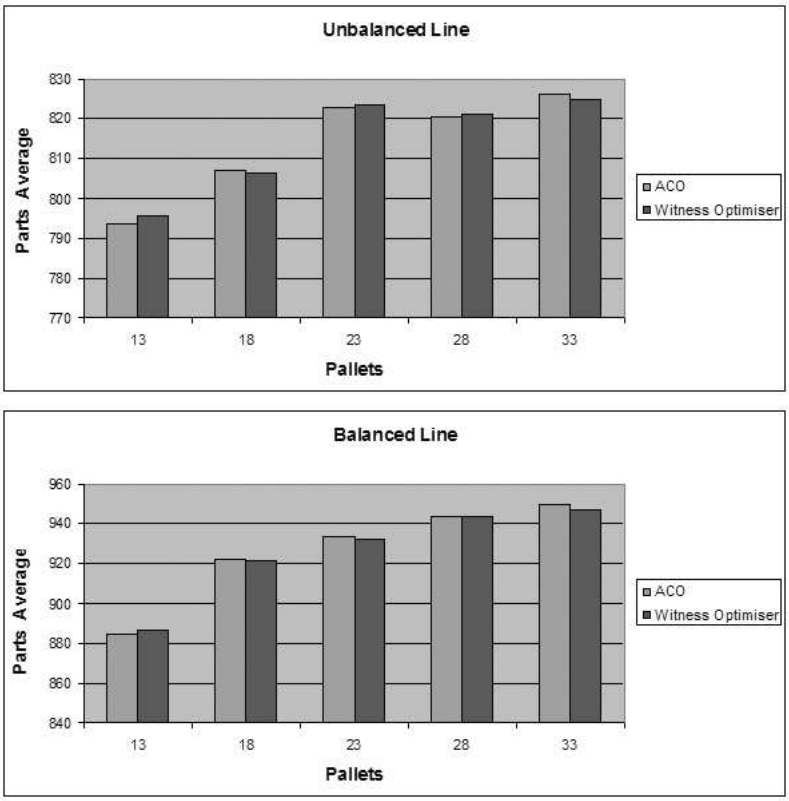


Fig. 5. Best average production rates for the balanced and unbalanced line configurations

to be conclusive, but even so, it could be justifiably stated that the ACO algorithm has performed competitively.

6 Conclusion

This paper has presented a new solution method for the buffer allocation problem in asynchronous assembly lines. The proposed solution incorporates both simulation and optimisation components: a specially developed stochastic ant colony algorithm is integrated with a simulation model of a closed-loop assembly line.

The main contributions of the proposed algorithm can be summarised as follows.

- It produces optimal or near-optimal solutions in a reasonable amount of computer time for which upper bounds can be established. In the case of large lines, solution quality may have to be traded off against time.
- With minor adaptations, it can cope with different line topologies, lengths and limits on buffer capacities to be distributed.
- It takes into account the main characteristics of the buffer allocation problem, i.e. discreteness of the buffer sizes, presence of multiple local optima and lack of an explicit objective function to evaluate production rate with respect to the buffer sizes.

- The algorithm is capable of overcoming local optima to find a global optimal solution.
- It is a stochastic algorithm: the generation of solutions is randomised to an extent. It probabilistically maps the solution space of the problem.

Future research could incorporate further testing, including more experimental runs and a wider array of alternative algorithms. Further experimental work would better establish the validity of the algorithm's performance. Additionally, the convergence properties of the algorithm could be studied and compared against the experimental findings.

References

Bulgak, A.A., P.D. Diwan, and B. Inozu. 1995. Buffer size optimization in asynchronous assembly systems using genetic algorithms. *Computers in Industrial Engineering Vol. 28. No. 2. pp. 309-322.*

Gershwin, S.B. 1994. Manufacturing Systems Engineering. *Prentice Hall, New Jersey.*

Papadopoulos, H.T. and C. Heavey. 1996. Queueing theory in manufacturing systems analysis and design: A classification of models for production and transfer lines. *European Journal of Operational Research. 92: 1-27.*

Altiok, T. 1997. Performance Analysis of Manufacturing Systems. *Springer-Verlag, New York.*

Swisher, J.R., P.D. Hyden, H.J. Sheldon and L.W. Schruben. A survey of simulation optimization techniques and procedures. *Proceedings of the 2000 Winter Simulation Conference.*

Fu, M.C. 2002. Optimization for Simulation: Theory vs. Practice. *INFORMS Journal on Computing Vol. 14. No. 3. pp.192-215.*

Blum, C. and A. Roli. 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys Vol. 35. No. 3. pp. 268-308.*

Spinellis, T.P. and Papadopoulos, C.T. 1999. Production line buffer allocation: genetic algorithms versus simulated annealing. *Second International Aegean Conference on the Analysis and Modelling of Manufacturing Systems, pp. 89-101.*

Spinellis, T.P. and Papadopoulos, C.T. 2000. A simulated annealing approach for buffer allocation in reliable production lines. *Annals of Operations Research 93. pp. 373 -384.*

Dorigo, M. and G. Di Caro. 1999. Ant algorithms for discrete optimization. *Artificial Life 5: 137-172.*

Gutjahr, W. 2004. S-ACO: an ant-based approach to combinatorial optimization under uncertainty. *Ants Workshop, pp. 238-249.*

Vitanov, I.V. 2006. A simulation-based ant colony algorithm for assembly line buffer allocation. *MSc thesis, Cranfield University.*

Particle Swarm Optimization and Genetic Algorithms for Portfolio Selection: a Comparison

L. Mous, V.A.F. Dallagnol, W. Cheung, and J. van den Berg

Econometric Institute, Faculty of Economics, Erasmus University Rotterdam
Room H10-19, P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
email: jvandenberg@few.eur.nl

Abstract. *The working of two nature inspired computing algorithms, Particle Swarm Optimization (PSO) and Genetic Algorithms (GA), is compared in case of solving a constrained portfolio optimization problem. Three strategies to handle the constraints were implemented for PSO: bumping, amnesia and random positioning. For GA, two selection operators, roulette wheel and tournament, have been tried and for handling the constraints two crossover operators, basic crossover and arithmetic crossover, have been experimented with. It is shown that both PSO and GA find solutions close to the optimal one where PSO appears to converge much faster than GA. It further turned out that the quality of the solution found by GA is somewhat better than that of PSO.*

1 Introduction

The problem presented in this research is that of an investor who has to choose (a) in which assets to invest and (b) how much in each of these assets. In other words: how can (s)he get the optimal portfolio? To answer this question, we need to define what an optimal portfolio is. The Markowitz' model ([1], [2]) assumes that investors only care about the mean and the variance of a portfolio. Samuelson [3] has defended this paradigm by showing that moments higher than the second are much less important than the first two moments. He also stated that for the investors' welfare, the variance must be considered as important as the mean of the returns. A portfolio containing multiple assets reduces the overall risk by diversifying away the idiosyncratic risk.

Other methods were developed as an alternative to the mean-variance framework. Konno and Yamazaki [4], for example, proposed the Mean Absolute Deviation model where the absolute deviation is used as a risk measure, hereby transforming the quadratic optimization problem into a linear problem. Instead of looking at variance as risk, Stanislav Uryasev [5] used Conditional Value-at-Risk (CVAR) in his approach to find the optimal portfolio. Young [6] addressed the issue by making use of minimum return as a measure of risk rather than variance (Minimax portfolio).

To solve the Markowitz' optimization problem, quadratic programming (QP) has often been used. However, if the problem includes a large number of assets or constraints, finding the best solution suffers from combinatorial explosion. In these cases, several heuristic approaches can and have been used including Genetic Algorithms(GAs) [7] and Particle Swarm Optimization(PSO) [8]. YusenXia et al. [9], for example, used a Genetic Algorithms for solving the mean-variance optimization problem with transaction costs. Chang et al. [10]

focused on calculating the mean-variance frontier with the added constraint of a portfolio only holding a limited number of assets. They have used three heuristics to solve this problem, including a Genetic Algorithm. Finally, Kendall [11] maximized the Sharpe ratio using Particle Swarm Optimization, but for only a very limited number of assets.

The objective of this paper is to compare the performance of PSO and the GA methods on the constrained portfolio optimization problem in case of having a large number of assets. As an extra constraint it is here imposed that no short selling is allowed. Furthermore, we will not pay attention to the problem of how to find the best estimates for the expected returns of each asset and the covariances between the assets. Instead, we shall simply assume that the estimates used are correct. The remainder of this paper is organized as follows. Section 2 gives a brief description about the theory of portfolio optimization. In the following sections, the focus is on heuristic methods. Section 3 presents the Particle Swarm Optimization method and section 4 discusses the use of Genetic Algorithms for solving the constrained portfolio optimization problem. Section 5 explains the experimental setup and summarizes the empirical results obtained. Finally, section 6 concludes this paper.

2 Portfolio Optimization

Efficient Portfolios Modern Portfolio Theory (MPT) assumes that investors are rational and risk adverse, meaning that for a given level of risk, investors prefer higher returns to lower returns. It is standard to measure risk in terms of the *standard deviation* (σ) of the return on an investment. Assuming riskless arbitrage opportunities do not exist, the investors would like to invest in an efficient portfolio, that is, one in which there is no other portfolio that offers a greater return with the same or less risk, or less risk with the same or greater return. When combining different assets into one portfolio, it is important to see how the movements of one asset are related to the movements of others. The degree to which several assets comove can be measured by the *covariance* (σ_{ij}). By using the variances of the individual assets and the covariances between them, it is possible to measure the overall risk (or volatility) of a combined portfolio. Investors are interested in the set of efficient portfolios all of which having minimal risk for different levels of returns. Together they form the *mean-variance efficient frontier*. Figure 1 illustrates a typical efficient frontier. Calculating the portfolios on the frontier can be done by means of quadratic programming (QP). The corresponding optimization problem for finding the portfolio p with minimal risk given a desired return R_p can be stated as follows:

$$\min_{w_i, w_j} \sigma_p^2 = \sum_{i,j} w_i w_j \sigma_{ij}, \tag{1}$$

$$\text{subject to: } R_p = \sum_{i=1}^{N} w_i r_i, \ \sum_{i=1}^{N} w_i = 1, \text{ and } \forall i : 0 \leq w_i \leq 1. \tag{2}$$

Here, σ_p is the standard deviation of the returns of portfolio p, w_i is the weight of the investment in asset i, σ_{ij} is the covariance between the returns of investments i and j, R_p is the return of portfolio p, N is the number of assets, and r_i is the return of the investment in asset i. Restricting the weights to non-negative values implies that we do not allow 'short selling' [11].

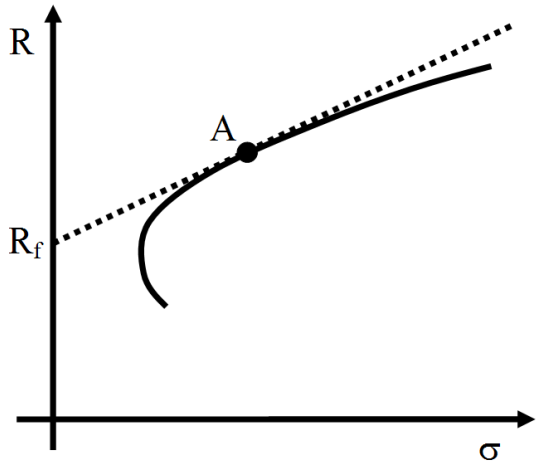


Fig. 1. Efficient frontier of portfolios consisting of only risky assets (upper part of continuous line) and the capital allocation line in case of combining with a riskless asset (dashed line).

Sharpe ratio So far we have looked at the optimal portfolio given a level of return. With an extra assumption, there is one portfolio on the mean-efficient frontier which is always better than the others. This portfolio is characterized by having the maximum Sharpe ratio. The Sharpe ratio S_p of portfolio p is defined as

$$S_p = \frac{R_p - R_f}{\sigma_p} = \frac{\sum_{i=1}^{N} w_i r_i - R_f}{\sqrt{\sum_{i,j} w_i w_j \sigma_{ij}}}, \tag{3}$$

where R_f is the risk-free rate. The Sharpe ratio uses excess returns $R_p - R_f$ to show how much extra return you may get for going from a riskless to a risky position. In order to get the optimal portfolio, the Sharpe ratio (3) should be maximized by varying the weights w_i where the two constraints $\sum_{i=1}^{N} w_i = 1$ and $\forall i : 0 \leq w_i \leq 1$ of (2) are fulfilled. Knowing the portfolio with maximal Sharpe ratio, investments can be set up consisting of both a riskless part (yielding a return R_f) and a risky part (yielding an expected return of the portfolio with maximal Sharpe ratio). The risk profile of the investor determines the way the investment is spread between the parts. All possible combinations of this type of investments have expected returns that lie on a 'capital allocation line' [12] as has been visualized in Figure 1. The point 'A' denotes the risky portfolio with the maximal Sharpe ratio.

In case of having a large numbers of assets available, the problem of calculating the portfolio with maximal Sharpe ratio is not a trivial task since large-scale constrained quadratic programming problems are very hard [4]. It becomes then worthwhile to consider heuristic methods including Particle Swarm Optimization (PSO) and Genetic Algorithms (GAs).

3 Particle Swarm Optimization

General Issues The Particle Swarm Optimization algorithm [8] is based on the behavior of fishes and birds, which collaboratively search a space to find food. Here, the search space

consists of the weight space defined by the investment weights w_i and the goal is to find a portfolio with optimal weights, i.e., the portfolio with highest Sharpe ratio. To get started, each particle gets a random initial position consisting of a weight vector (w_i) where each of the first $N-1$ weights is given a positive random value and where the weight of the last asset is simply determined conform $w_N = 1 - \sum_{i=1}^{N-1} w_i$. Given the weight vector (w_i) of a particle, the Sharpe ratio of the corresponding portfolio can be calculated according to (3).

To enable searching, the particles are also initialized with a random velocity, in a well spread manner. The velocity in an iteration is calculated by the following formula:

$$\mathbf{v}_i(t+1) = \alpha \mathbf{v}_i(t) + r_1(\mathbf{p}_i^{best} - \mathbf{p}_i(t)) + r_2(\mathbf{g}^{best} - \mathbf{p}_i(t)), \tag{4}$$

where $\mathbf{v}_i(t)$ is the velocity vector of particle i at time t, α is a weight which gives the particle momentum, r_1 and r_2 are random numbers, $\mathbf{p}_i^{best}$ is the best position of particle i in all iterations (i.e., the position of particle i with weights corresponding to the portfolio with so far highest Sharpe ratio), $\mathbf{g}^{best}$ is the best position of the entire population in all iterations, and $\mathbf{p}_i(t)$ is the position of particle i at time t. The velocity is determined independently for each dimension and each asset. In general, α reduces as the number of iterations increases, and is here defined as

$$\alpha = \alpha_{max} - \frac{\alpha_{max} - \alpha_{min}}{IT} \cdot it, \tag{5}$$

with α_{max} and α_{min} being the value of w for the first and last iterations, respectively, IT being the total number of iterations, and it the number of the present iteration. In this implementation, the following values of α_{max} and α_{min} were found, through experimentation, to be the best: $\alpha_{max} = 0.9, \alpha_{min} = 0.4$.

Constrained PSO As has been mentioned above, not allowing short selling means that the weights of the assets in the portfolio cannot be negative. In case of N assets, the feasible solutions space is delimited by hyperplanes with equations $w_1 = 0, \ldots, w_{N-1} = 0, \sum_{i=1}^{N-1} w_i = 1$. In the specific case with 3 assets, the hyperplanes are reduced to the lines $w_1 = 0, w_2 = 0, w_1 + w_2 = 1$, the corresponding triangular area of feasible weights is visualized in figure 2. When applying the no short sales constraint, the optimal solution of the constrained optimization problem will then lie on the boundary of the feasible solutions space, in the point represented by a black star in the picture. Because of this and of the fact that particles have a momentum, it is very likely that the particles will eventually cross the boundaries. There are different strategies to make sure that the particles still abide the non-negative constraint and each has its advantages and disadvantages. According to [13] the conventional methods to make sure that all the particles stay within the feasible space are 'bumping' or 'random positioning'.

Bumping resembles the effect of a bird hitting a window. As the particle reaches the boundary of the feasible space, it 'bumps'. The particle stops on the edge of the feasible space and loses all of its velocity. Bumping can be implemented as follows. Let the position vector of a particle at $t = 1$ be $\mathbf{p}(1) = (w_1(1), \ldots, w_N(1))'$ and the position at $t = 2$ be $\mathbf{p}(2) = (w_1(2), \ldots, w_N(2))'$ where $\mathbf{p}(2)$ is outside the feasible solutions space with some weight(s) $w_i(2) < 0$. Then, to avoid that the particle enters the infeasible space, it is stopped at $\mathbf{p}"(2)$ defined by the weights $w_i"(2)$ which are recalculated according to

$$w_i"(2) = w_i(1) + v_i \cdot \frac{w_i(1)}{w_i(1) - w_i(2)} \qquad \text{for } i = 1, ..., N-1, \tag{6}$$

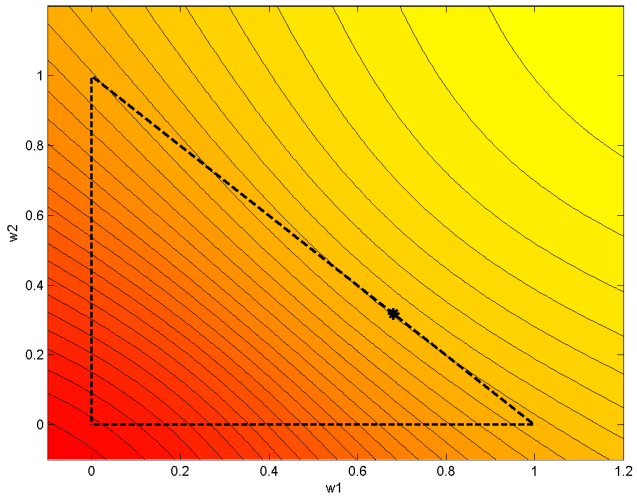


Fig. 2. Portfolio optimization with 3 assets, with optimal solution near the boundary. The contour lines show portfolios with the same Sharpe ratio. Darker (red) regions denote portfolios with lower Sharpe ratios and brighter (yellow) areas correspond to portfolios with higher Sharpe ratios. The star in black is the optimal portfolio given the constraint of no short selling.

where v_i is the velocity component i of the particle. The weight w_N"(2) is calculated by w_N"(2) $= 1 - \sum_{i=1}^{N-1} w_i$"(2), as usual. After bumping, in the next iteration the particle will gain velocity again, starting from velocity zero and applying (4). If the $\mathbf{g}^{best}$ is near the edge of feasible space, then bumping makes sure the particles remain near the edge and thus near the $\mathbf{g}^{best}$. However the particles may get 'trapped' at the current $\mathbf{g}^{best}$, because they stop there all the time. Because of that and the loss of velocity, the entire search space of all particles is reduced and premature convergence may occur. Thus the particles may get stuck in a sub-optimal solution.

Random positioning simply changes the negative weight into a random, feasible weight. The advantage of this with respect to bumping is the fact that the search space is not reduced. In fact the search space is expanded, because the particle is placed in an entirely different area. The premature convergence, which may occur with bumping, does not occur here. However the opposite may be true for random positioning: there will no convergence to the global optimum at all. If the optimal solution is near the edge, then a particle approaching it may be thrown back into a different and worse area. It will then fly back towards the optimum, touch the edge, and get thrown back again. Thus the particle may get stuck in a loop and will never be able to come to a stop at the best solution.

Hu and Eberhart [14] come with a different strategy, henceforth called *amnesia*. With the initialization all the particles are feasible solutions. But during the iterations the particles are allowed to cross the boundaries and fly into the infeasible space. However the particles will not remember the value of the objective function of the solutions found in the infeasible space and if they find a better solution, it will not be recorded neither as $\mathbf{p}_i^{best}$ nor as $\mathbf{g}^{best}$. Because of this, $\mathbf{p}_i^{best}$ and $\mathbf{g}^{best}$ will always lie inside the feasible space and thus the

particles will be attracted back there. Amnesia solves the problems of bumping and random positioning: the size of the search space is not reduced, while the particles stay around the edge and the current $\mathbf{g}^{best}$. Convergence will therefore take place, but it will not easily be premature convergence. A downside of amnesia may be the fact that the particles do 'waste time' flying in infeasible space, which in some cases may result in an inefficient search.

4 Genetic Algorithms

Besides the PSO, GAs [7] are also used to solve this portfolio optimization problem. Here, the *chromosomes* are implemented in the same way as the solutions or particles in PSO, i.e., as strings of real-valued numbers, with each number representing the weight of an asset. This is called real-coding [15], and allows the algorithm to search for a solution in a continuous space, which seems more natural in the portfolio optimization problem. And because the same encoding is used for PSO, the performance can be compared more easily.

Design of the GA operators 'selection', 'crossover', and 'mutation' is a major issue in a GA implementation. Two different *selection* methods were implemented: the tournament selection and the roulette wheel selection [16]. The other two operators should be implemented in a way that the two constraints $\sum_{i=1}^{N} w_i = 1$ and $\forall i : 0 \leq w_i \leq 1$ of (2) remain fulfilled. Two different crossover operators were implemented: the 'simple arithmetic crossover' and the 'whole arithmetic crossover'. Both are crossover operators commonly used for real-coded GA [17]. Each crossover involves two parents, who were selected via the selection operator, and the crossover combines their genes to produce two offsprings. *Simple arithmetic crossover* resembles the basic crossover for binary solutions. First a random point is chosen on the parent's chromosomes. Then the crossover swaps the solutions after that point. After this swap a constraint may have been violated: the weights no longer sum up to one. Because of this, the weights must be normalized after the swap which introduces an unintended extra impact on the weights. It increases the randomness and thus, increases the exploration of the space and reduces the exploitation of good solutions. However, as a new offspring's weights can sum up to no more than 2, normalizing the weights will in the worse case result in dividing the weights by 2, which seems acceptable. The nonnegative constraint however will never be violated with this crossover, because nothing is subtracted from the weights. The *whole arithmetic crossover* is more adjusted to using real numbers instead of binary ones. Here the first offspring consists of a fraction p_1 of parent 1 and $1 - p_1$ from parent 2, with $p_1 \in [0, 1]$. The second offspring will be the opposite, being composed as a fraction $1 - p_1$ of parent 1 and p_1 of parent 2. The advantage of this crossover is that it automatically holds to both constraints: the weights stay positive and sum up to one. The final operator is *mutation*. To make sure that afterwards the weights of a solution still add up to one, the mutation operator was designed so that it randomly chooses two weights. Then an equal random amount is subtracted from one weight and added to the other one. Thus the sum of all weights is still one. Unfortunately a substraction may cause a weight to become negative. If this is the case then the weight's value is set to zero and the weights are normalized.

After good solutions have been selected, they are combined via the crossover to produce their offspring and after a possible mutation, the population of the next generation is formed. Since we wish to preserve the very best solutions across generations, the elitism strategy is included. Elitism means that the best n solutions of a generation (in our case, $n = 1$) should be transmitted untouched to the next generation. The parameters for GA concern

two probabilities. After a solution is selected, cross-over may be performed on it and a mutation after that. But the key-word here is *may*. After several tests, the probability of a crossover occurring was chosen to be 0.8 and of a mutation 0.01.

5 Experimental Setup and Results

As mentioned in the introduction, this study does not focus on getting correct estimates for the expected return and (co)variances. Because of this, we simply used the arithmetic mean and the (co)variance of historical data as expected return and risk, respectively. The historical data needed for performing these calculations, have been collected from the Datastream database. Since we want to look at how well the algorithms perform with respect to a large number of assets, we have decided to use the stocks from the Standard & Poor's 500. We have selected the monthly values of 493 stocks, in the period from February 2004 until February 2006. This results in 26 values per asset. Some of the stocks appeared to have, on average, a negative return and because of this, the algorithms should never pick them. They were still kept in the database in order to better test the performance of the algorithms.

First results The *first test* ran was designed to verify the 'consistency' of the algorithm defined as the ability of the algorithm to always find the optimal solution. This measure is needed because both algorithms have random components, and perhaps one simply performed better in a single attempt because it was lucky. The presence of assets with negative returns makes the problem harder because the best solutions may be near the boundaries of the feasible space. For each given number of assets in the portfolio, 5 sets of assets were randomly picked. Each strategy was run 5 times over these sets, to try to eliminate the influence of a lucky initial solution. To allow for a fair comparison, all the runs were made keeping the number of particles/chromosomes (for PSO/GA) constant and equal to 25 and in the next experiment equal to 50 (see Table 1). Each algorithm was run for 1000 iterations which was supposed to be a number big enough. Based on the results obtained in this way, the average number $\overline{N_{it}}$ of iterations that a certain strategy needs to find a solution within 0.1% of the best Sharpe Ratio found on that particular run, was calculated. This number is an indicator for the speed of convergence to the neighborhood of the best solution. To test the consistency of this result, the standard deviation σ_N was also calculated. Next, the average error $\overline{\varepsilon_{SR}}$ was calculated between (i) the best Sharpe ratios found by the algorithms in the particular runs and (ii) the best Sharpe ratio found by the different strategies on all runs, for a particular set of assets, together with the standard deviation of this measure σ_ε. This allows to understand whether the algorithm is good to find the best solution.

The *second test* was designed to examine the speed of the different strategies. As the speed of the algorithm is related to the number of iterations executed, the algorithms were run with a fixed number of iterations equal to $\overline{N_{it}} + 2\sigma_N$ (extracted from Table 1). Considering a normal distribution of N_{it}, it means that approximately 97.7% of the time the algorithm will have enough iterations to arrive within 0.1% of the best Sharpe Ratio that would be found in that particular run. For each given number of assets in portfolio, 5 runs over each of the 5 randomly picked sets of stocks were executed, like done in the previous test, and the average time of execution (in seconds) of the algorithm ($\bar{t}$) and standard deviation of this time (σ_t) were calculated. The results are presented in Table 2.

Table 1. Comparison of the *consistency* of PSO and GA algorithms for solving the portfolio optimization problem, running with 25 respectively 50 particles/chromosomes. N_a is the number of assets included in the portfolio; $\overline{N_{it}}$ is the average number of iterations that the algorithm took to find a solution within 0.1% of the best SR found; σ_N is the standard deviation of this measure; $\overline{\varepsilon_{SR}}$ is the average error between the SR found by the algorithm in the different runs and the best SR found overall runs; and σ_ε is the standard deviation of these errors

		25 particles				50 particles			
N_a	Algorithm	$\overline{N_{it}}$	σ_N	$\overline{\varepsilon_{SR}}$	σ_ε	$\overline{N_{it}}$	σ_N	$\overline{\varepsilon_{SR}}$	σ_ε
	PSO Bumping	24	24	-0.12%	0.25%	16	14	-0.03%	0.04%
	PSO Amnesia	117	50	-0.40%	0.69%	79	65	-0.05%	0.13%
	PSO Random	26	14	-0.12%	0.21%	15	15	-0.07%	0.24%
5	GA Roul./Basic	121	120	-0.80%	3.98%	60	116	-1.86%	4.92%
	GA Tourn./Basic	127	112	-0.00%	0.00%	36	25	-0.01%	0.01%
	GA Roul./Arith.	346	176	-0.80%	3.92%	314	282	-3.08%	4.71%
	GA Tourn./Arith.	251	134	-0.00%	0.00%	129	101	-0.01%	0.01%
	PSO Bumping	101	52	-1.43%	2.06%	55	49	-0.21%	0.42%
	PSO Amnesia	217	51	-2.01%	2.04%	144	50	-0.39%	0.82%
	PSO Random	118	70	-2.13%	2.23%	66	48	-0.27%	0.74%
10	GA Roul./Basic	285	187	-3.91%	10.27%	247	111	-0.00%	0.00%
	GA Tourn./Basic	483	193	-0.04%	0.07%	281	215	-0.02%	0.02%
	GA Roul./Arith.	827	128	-0.72%	0.43%	853	104	-1.04%	0.50%
	GA Tourn./Arith.	756	186	-0.37%	0.53%	624	182	-0.06%	0.05%
	PSO Bumping	188	87	-9.38%	5.39%	171	87	-2.71%	3.29%
	PSO Amnesia	290	41	-10.09%	5.06%	301	46	-3.52%	3.29%
	PSO Random	170	77	-9.22%	5.86%	204	92	-3.06%	3.46%
20	GA Roul./Basic	808	124	-0.24%	0.37%	661	231	-0.04%	0.05%
	GA Tourn./Basic	835	180	-1.14%	1.03%	689	242	-0.31%	0.31%
	GA Roul./Arith.	884	117	-4.38%	2.04%	921	64	-5.79%	2.78%
	GA Tourn./Arith.	895	76	-2.10%	1.38%	869	112	-0.95%	0.82%
	PSO Bumping	209	109	-20.04%	8.42%	220	88	-9.75%	6.48%
	PSO Amnesia	372	40	-22.04%	8.48%	380	44	-12.79%	6.50%
	PSO Random	180	84	-16.82%	5.99%	241	93	-9.88%	6.32%
30	GA Roul./Basic	965	35	-1.90%	2.14%	884	78	-0.51%	0.62%
	GA Tourn./Basic	897	101	-3.75%	4.49%	873	121	-1.05%	0.96%
	GA Roul./Arith.	894	197	-10.81%	9.24%	947	40	-9.61%	2.35%
	GA Tourn./Arith.	941	49	-7.80%	4.72%	930	77	-3.79%	1.98%
	PSO Bumping	210	72	-30.18%	7.98%	243	75	-22.43%	9.26%
	PSO Amnesia	479	36	-33.73%	9.43%	485	27	-30.63%	9.84%
	PSO Random	200	99	-28.45%	10.73%	278	105	-20.97%	9.86%
50	GA Roul./Basic	961	38	-5.37%	4.66%	965	29	-2.06%	2.49%
	GA Tourn./Basic	937	55	-8.87%	5.51%	877	122	-3.33%	2.85%
	GA Roul./Arith.	948	41	-14.62%	4.06%	933	53	-18.39%	4.22%
	GA Tourn./Arith.	927	71	-11.47%	3.41%	961	37	-8.74%	3.33%

Selecting best PSO and GA Before GA and PSO are compared, the individual strategies of the algorithms are first looked at. *PSO* had three different strategies for handling the nonnegativity constraint. When looking at tables 1 and 2, a first observation is that the strategy of letting the particles fly into infeasible space areas is not a very good strategy. In

Table 2. Comparison of the *speed* of the PSO and GA algorithms for solving the portfolio optimization problem. N_a is the number of assets included in the portfolio; Algorithm identifies to which algorithm the listed results refer to; N_{it} is the fixed number of iterations used to solve the problem; $\bar{t}$ is the average time in seconds to solve the optimization problem; and σ_t is the standard deviation in seconds of the time to solve the optimization problem

N_a	Algorithm	N_{it}	$\bar{t}$ (s)	σ_t (s)
	PSO Bumping	44	0.20	0.118
	PSO Amnesia	210	0.78	0.005
	PSO Random	45	0.18	0.008
5	GA Roul./Basic	293	2.94	0.025
	GA Tourn./Basic	86	0.35	0.010
	GA Roul./Arith.	878	7.76	0.209
	GA Tourn./Arith	332	1.12	0.014
	PSO Bumping	153	0.78	0.010
	PSO Amnesia	243	1.19	0.014
	PSO Random	163	0.83	0.011
10	GA Roul./Basic	468	5.21	0.018
	GA Tourn./Basic	711	3.58	0.023
	GA Roul./Arith.	1062	10.52	0.041
	GA Tourn./Arith	988	4.37	0.030
	PSO Bumping	345	3.02	0.025
	PSO Amnesia	394	3.34	0.029
	PSO Random	389	3.39	0.029
20	GA Roul./Basic	1124	16.73	0.203
	GA Tourn./Basic	1172	10.09	0.074
	GA Roul./Arith.	1050	14.24	0.061
	GA Tourn./Arith	1092	8.71	0.073
	PSO Bumping	396	5.53	0.030
	PSO Amnesia	469	6.41	0.026
	PSO Random	427	5.95	0.027
30	GA Roul./Basic	1041	20.93	0.052
	GA Tourn./Basic	1116	15.35	0.048
	GA Roul./Arith.	1028	19.32	0.055
	GA Tourn./Arith	1084	14.21	0.049
	PSO Bumping	394	14.78	0.460
	PSO Amnesia	541	19.99	0.625
	PSO Random	488	18.37	0.664
50	GA Roul./Basic	1024	44.03	0.221
	GA Tourn./Basic	1121	40.44	0.076
	GA Roul./Arith.	1039	43.24	0.269
	GA Tourn./Arith	1035	36.63	0.064

fact it was not uncommon for the best solution to stay the same for 300 iterations before being improved by the algorithm. So clearly the Amnesia strategy for PSO should not be used for this problem.

This observation leaves Bumping or Random Positioning. It turned out that, according to Table 1 in case of using 50 particles, Bumping is more consistent in obtaining the best overall Sharpe ratio than Random Positioning. This is however not the case for 50 assets, so it

is possible that Bumping is only better for smaller number of assets. A similar behavior might be expected in case of using less particles. It indeed turned out that Random Positioning reaches a better Sharpe ratio for 20, 30 and 50 assets. This behavior may be due to the fact that Random Positioning has better exploratory properties, which is needed for finding optimal solutions of complex problems, than Bumping. It seems therefore true that Bumping is (only) better if there are enough particles or not too many assets to choose from. However, the performance difference between Random Positioning and Bumping is not clear enough to say that for certain. It was also speculated that Random Positioning might be often unable to reach the global optimum within the time available. For example, if the solution lies near the boundary, at every time a particle reaches the boundary it is thrown back into the feasible space, away from the most promising area. In turned out that Random Positioning indeed requires slightly more iterations than Bumping. So, to summarize, the PSO algorithms with Bumping and Random Positioning perform much better than that with the Amnesia strategy while the quality differences in performance with between the first two are small. Since the PSO with Bumping is consistently faster than that with Random Positioning we have selected that one to use for comparison with the GA algorithm.

The *Genetic Algorithm* had two different selection strategies and two different crossover operators to choose from, all together four different strategies. First the resulting Sharpe ratio is looked at. It turned out that the basic crossover strategies usually beat the arithmetic ones, regardless of which selection method used. So, the arithmetic strategies could be ignored from now on. Looking at the results found by the GA with basic crossover using different selection strategies (roulette wheel or tournament) we notice that for some cases, the tournament selection operator does better, and for other cases, the roulette wheel won. On the other hand, the tournament/basic strategy generally runs somewhat faster than the roulette/basic strategy. That is why we have selected the GA with the tournament/basic strategy as the best GA for this problem.

Comparing best PSO with best GA Finally we make a comparison between the PSO algorithm using Bumping and the GA with tournament/basic strategy. It turned out that PSO and GA both need a similar amount of time to calculate one iteration, and that PSO consistently requires much less iterations, and therefore takes much less time. On the other hand, PSO's resulting Sharpe ratio's are not as good as those found by the GA. Therefore it seems that, while PSO is faster in getting near the global optimum, it does not have GA's finesse in getting the precise optimal values. This difference can be seen in Figures 3 and 4. These figures show a typical progress of the 50 particles/chromosomes for the problem of 50 assets. Again the x-axis of both graphs show that Bumping requires much less time.

We further observe that the best solution of Bumping usually increases steadily, while the best one of GA usually takes small steps and sometimes, suddenly, a very large step (for example around 450 iterations). Apparently, PSO/Bumping's particles are usually steadily flying towards the global optimum, while the GA chromosomes may sometimes do big steps in the right direction due to a lucky search combination. These sudden improvements can also be observed in the performance of the worst solution of the GA. Probably due to GA's operators, some GA solutions are sometimes thrown away from the interesting area and end up somewhere in a completely different area. Because of that the value of the worst solution may have big fluctuations.

A final observation with respect to the performances of PSO and GA concerns the differences between the best Sharpe ratio and the average one found. With GA, the two

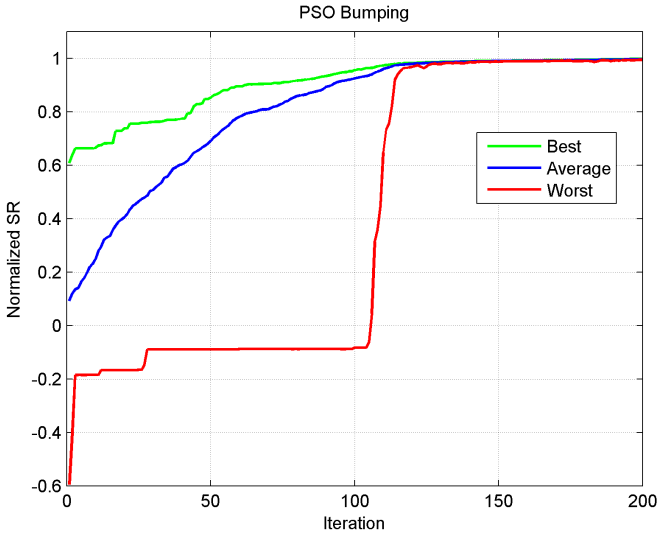


Fig. 3. Evolution of the solutions found by the PSO algorithm with bumping strategy, running with 50 particles, in a problem with 50 assets.

lines are very close to each other, indicating that the population consists of similar solutions and Sharpe ratios. But with PSO, initially, there is a big difference between the average and best solution indicating that, generally, the particles are spread further across the solution space. Only in the end, when all the particles reach the interesting area, the difference between the average and the best value decreases. Since in the end, the particles of PSO are focusing on just one promising area and all fly directly towards it, PSO requires less time than GA. The GA on the other hand seems to be better in continuously exploring interesting areas and, because of this, in producing better final solutions: please see the values of $\overline{\varepsilon_{SR}}$ and σ_ε in Table 1. However, the differences in time and in values of the Sharpe ratio are not so large that a clear winner of the two algorithms can be announced. Which method is better therefore depends on the context of the problem: does one need a fast method with a good result, or does one have more time available and demands a great(er) result?

6 Conclusions

The performance of two systematic random search algorithms, Particle Swarm Optimization and Genetic Algorithms, applied to a constrained portfolio optimization problem, was shown and analyzed. Both algorithms are inspired by phenomena occurring in Nature, but the specific mechanics are different. Several strategies to handle the constraints were implemented and the results were compared. For PSO, the strategies bumping, amnesia and random positioning have been implemented. For GA, the constraints were handled in the selection (roulette wheel and tournament) and crossover (basic crossover and arithmetic crossover) operators.

It has been shown that both PSO and GA find solutions close to the optimal one where PSO appears to converge much faster than GA. It further turned out that the quality of

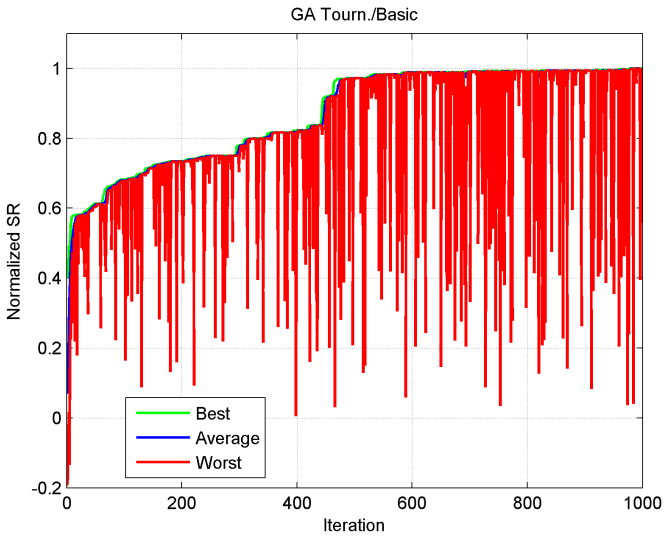


Fig. 4. Evolution of the solutions found by the GA algorithm with tournament selection and basic crossover, running with 50 chromosomes, in a problem with 50 assets.

the solution found by GA is somewhat better than that of PSO. The results also show that both algorithms may be used for this constrained optimization problem but specific parameters (e.g. number of particles and number of iterations/generations) must be tuned ad hoc. Future research should reveal whether this behavior of the two types of algorithms is systematic by applying them to a series of other (constrained) optimization problems.

References

1. Markowitz, H.M.: Portfolio selection. Journal of Finance **7** (1952) 77 – 91
2. Markowitz, H.M.: Portfolio Selection: efficient diversification of investments. John Wiley & Sons Inc, New York (1959)
3. Samuelson, P.A.: The fundamental approximation theorem of portfolio analysis in terms of means, variances, and higher moments. Review of Economic Studies **37** (1970)
4. Konno, H., Yamazaki, H.: Mean-absolute deviation portfolio optimization model and its applications to tokyo stock market. Management Science **37**(5) (1991) 519 – 531
5. Uryasev, S.: Conditional value-at-risk: Optimization algorithms and application. Financial Engineering News (14) (2000)
6. Young, M.R.: A minimax portfolio selection rule with linear programming solution. Management Science (44) (1998) 673–683
7. Holland, J.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)
8. Eberhart, R., Kennedy, J.: Particle swarm optimisation. In: Proceedings of the IEEE International Conference on Neural Networks (ICNN'95). (1995) 1942–1948
9. Xia, Y., Liu, B., Wang, S., Lai, K.K.: A model for portfolio selection with order of expected returns. Computers & Operations Research (27) (2000) 409–422

10. Chang, T.J., Meade, N., Beasley, J., Sharaiha, Y.: Heuristics for cardinality constrained portfolio optimisation. Computers & Operations Research (27) (2000) 1271–1302
11. Kendall, G., Su, Y.: A particle swarm optimisation approach in the construction of optimal risky portfolios. Proceedings of the 23rd IASTED International Multi-Conference Artificial Intelligence and Applications (2005)
12. Sharpe, W.F.: The sharpe ratio. Journal of Portfolio Management (fall 1994)
13. Zhang, W.J., Xiao-Feng Xie, D.C.B.: Handling boundary constraints for numerical optimization by particle swarm flying in periodic search space. Congress on Evolutionary Computation (CEC), Oregon, USA (2004) 2307–2311
14. Hu, X., Eberhart, R.: Solving constrained nonlinear optimization problems with particle swarm optimization. Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), Orlando, USA **5** (2002)
15. Arumugam, M.S., Rao, M.: Novel hybrid approaches for real coded genetic algorithm to compute the optimal control of a single stage hybrid manufacturing systems. International Journal of Computational Intelligence **1**(3) (2004) 231–249
16. Rawlins, G.J.: Foundations of genetic algorithms. Morgan Kaufmann Publishers (1991)
17. Blanco, A., Delgado, M., M.C.Pegalajar: A real-coded genetic algorithm for training recurrent neural networks. Neural Networks **14**(1) (2001) 93–105

Automated Self-Assembly Programming Paradigm: A Particle Swarm Realization

Lin Li, Natalio Krasnogor, and Jon Garibaldi

ASAP Group
School of Computer Science and Information Technology
University of Nottingham
Nottingham
UK
{lxl,nxk,jmg}@cs.nott.ac.uk

Abstract. *In our previous work [14], we introduced Automated Self-Assembly Programming Paradigm ($ASAP^2$). We investigated how external environment settings affect population diversity and software self-assembly efficiency. In this paper, we introduce a mathematical model for $ASAP^2$ that allows us to predict time to equilibrium and diversity of generated solutions. In addition, we extended the work and illustrate an improved methodology for $ASAP^2$ based on particle swarm optimization. We carry out the same set of experiments as we did before and compare it with the results obtained from the interpolation. Results indicate that an improvement in software self-assembly efficiency can be achieved.*

1 Introduction

Self-assembly is a process in which components autonomously assemble into a complex structure through statistical exploration of alternative configurations. Components interact with each other without the presence of central control mechanism, nor external interventions in this process. Natural examples of self-assembly systems exist which are both inorganic and organic. Take protein folding as an example, components are amino acids arranged in a sequential order and the assembled structure is the folded three-dimensional shape of the protein. For a mobile condensed phase such as liquid, self-assembling reactions are mediated by chemical forces such as hydrogen bonding and van der Waals forces [17], [19]. Robustness and versatility are some of the most important properties of self-assembling systems. These two advantages come from the fact that self-assembly systems are usually composed of a large number of components and only some are used to construct the final goal structure. If a certain part of the self-assembled system fails, other components can replace the failed parts to ensure the functionality of the system. Self-assembly is also a versatile process because a goal structure can usually be constructed using different combinations of components, and this makes it prone to alternative configurations [8].

In our previous work [14], an Automated Self-Assembly Programming Paradigm ($ASAP^2$) inspired by natural and artificial self-assembly systems was presented. In $ASAP^2$, a set of human made software components are collected in software repositories and later integrated through self-assembly into a goal-specific software architecture aimed at satisfying certain pre-specified tasks. We investigated and discovered different factors that influence the efficiency of software self-assembly and the diversity of the generated programs using *unguided*

self-assembly. In this paper, we extend our previous work [14] by introducing a mathematical model to predict software self-assembly efficiency and population diversity for $ASAP^2$. To the best of our knowledge, similar predictive capacity has not yet been achieved in GP. Although genetic programming is one of the most popular methodologies for automated program synthesis [1] and has been applied to a wide range of problems [11], [10], [2], [3], we are interested in investigating a possible alternative to Genetic Programming.

It is our hope and long term goal that by designing an alternative paradigm for automatic program synthesis from the bottom up while keeping an eye on predictive model, we would one day have a principled engineering methodology for $ASAP^2$. Thus, rather than natural selection as a metaphor for the automated construction of programs, we intend to focus instead on self-assembly, which could then lead to self-healing [16] and self-reconfigurable systems [18].

Swarm intelligence mimics the behaviour of natural creatures such as particle swarm and flying birds (i.e. a flock). Particle swarm optimization (PSO) was first introduced by Kennedy and Eberhart [6] and it is one of the most common techniques used in swarm intelligence. PSO is a population-based search algorithm in which each individual in the population represents a solution. Individuals in PSO have various properties, among them each individual has its own velocity. More importantly, PSO is in some way similar to software self-assembly that each individual in PSO is an autonomous agent that follows simple rules. However, individuals as a whole exhibit emergent behaviours similar to the real particle swarm. In [15], [5], fitter particles are recorded into a leader set. Non-leader particles randomly select a leader from the leader set and follows the leader. In [9], particles determine and change their neighbours dynamically according to the distances in each generation.

In this paper, we present a new software self-assembly system with dynamic leaders and neighbourhood inspired by the previous work in PSO [9], [15], [5]. With our newly extended system, we perform the same set of experiments done in [14]. And we use our obtained formulae for $ASAP^2$ to compare the experiment results with our swarm based system. The comparison shows a significant improvement in software self-assembly efficiency.

The paper is organized as the following: in section 2, we will briefly describe our previous work on $ASAP^2$. In section 3, we will explain how we perform experiments for $ASAP^2$ and we review our previous experimental results. In section 4, a predictive model to interpolate software self-efficiency against various environment parameters will be presented for the previous work on unguided software self-assembly. We will explain our extended system and demonstrate the experiment in section 5. In section 6, we will analyze experimental data obtained from our extended system and assess our predictive model on these data. And the last section is the conclusion.

2 Program gas and unguided software self-assembly

In [14], we presented $ASAP^2$ which is loosely based on the theory of ideal gases.

- A two dimensional pool that plays the role of a gas container is used for software self-assembly. Software components are placed into the pool and move randomly.
- The area of the pool is a free parameter of the model.
- Software components are linked to molecules. The probability of movement is a function of the temperature.
- The temperature is another free parameter of the model.

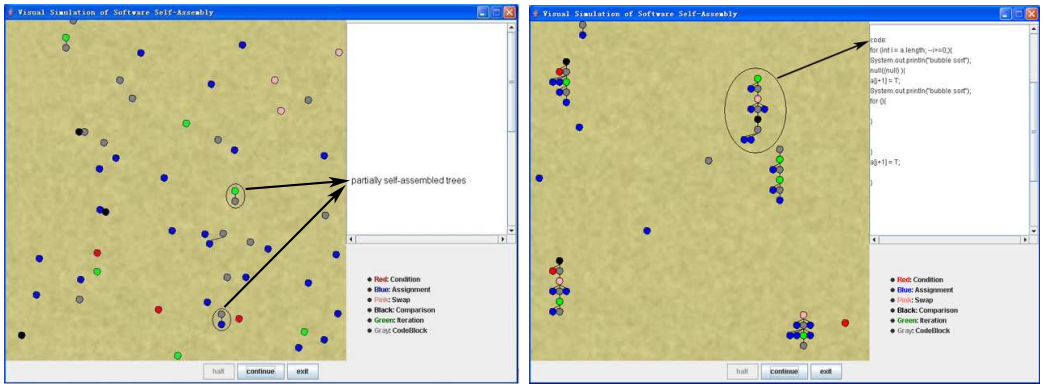
- Finally, the number n of components placed within the pool is the third and last free parameter.

We use the three parameters above in order to control the software self-assembly process. We investigated and discovered how these environment parameters can affect the efficiency of software self-assembly process and the diversity of the generated population. Because we are using ideal gas as a metaphor for $ASAP^2$, the well-known equation of states for ideal gases is used as a crude approximation to link the three free parameters described above to the dependent variable, i.e. the pressure on the walls of the pool exerted by the software components. The perfect gas has the following simple equation of state relating the pressure on the wall P, absolute temperature T, volume V and amount of molecules n:

$$PV = nRT, \tag{1}$$

where R is a constant value for the gas. Due to some major differences between program gas and perfect gas (as pointed out in [14]), we measured to what degree the equation holds for programme gases. Besides, we investigated how software self-assembly efficiency and diversity of the generated population is affected by the above environment parameters. Fig. 2 shows screenshots of $ASAP^2$, in which the colored circles represent software components.

In $ASAP^2$, a software component contains ports that play the role of binding sites. Ports can be divided into two categories: input ports and output ports, with a corresponding data type associated. We assume that an input port can only connect with an output port of the same data type to ensure only valid and meaningful configurations.



(a) (b)

Fig. 1. (a) Early stage of software self-assembly. (b) Latter stage of software self-assembly.

Software self-assembly starts by placing all the software components from a given software repository into the pool. In [14], software components move randomly in the pool with a probability which is a function of the temperature. When two components are close within a cut-off distance and their types match, they self-assemble into a larger aggregated structure. Otherwise, the two software components will be pushed away against each other. The left panel in Fig. 2(a) shows an early stage of software self-assembly in which few software

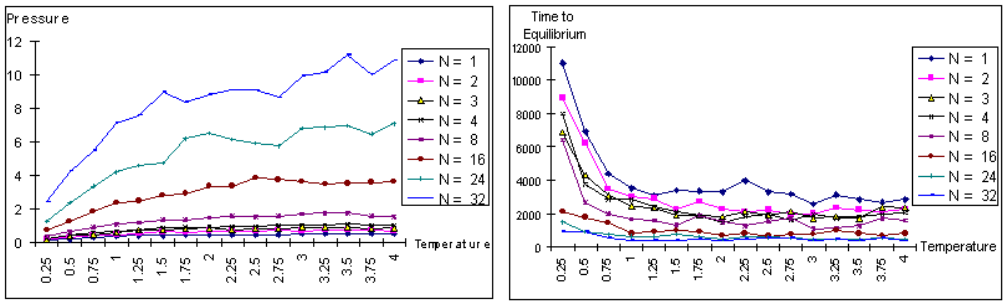
components are bound together. Fig. 2(b) shows a latter stage where more and more larger trees emerged. The simulation is deemed to have reached *equilibrium* when no more bindings can occur between the remaining (partial) self-assembled parse trees. Hence, equilibrium is the stopping condition for our simulations.

3 Review of previous results

We investigated in [14] to what degree Eq. 1 holds, i.e. the relationship between temperature (T), area (A), number of software components in the pool (N), pressure (P), time to equilibrium (t_ε) and diversity of the self-assembled trees at equilibrium (D_ε). More specifically, P is measured by counting the number of components that hit one of the four walls of the pool per unit of time. As mentioned above, T is a free parameter and it affects a component's moving probability, $p(M)$, where $p(M) = e^{-T}$. The area A of the pool and number of copies of each component, N, are the other two free parameter of the model.

Besides P we also measure "time to equilibrium" (t_ε), which records how long it takes the system to reach equilibrium, and diversity of the self-assembled parse tree classes (D_ε) at equilibrium. D_ε is used to assess the diversity of the emergent parse trees under different settings of N, A and T. For measuring diversity we consider that two trees belong to the same parse tree class if both their *structures* and *content* are identical [4].

We set up the experiments using a bubble sort program implemented in java as the source for software components. We represent this original bubble sort program into a parsing tree, and each node in this tree reprensents a software component. The original program is then cut in an *ad hoc* manner into 18 components and placed into a software repository. We run 20 replicas for each (A, T, N) triplet studied measuring the dependent variables. The ranges for A, T and N were: $A \in \{160k, 250k, 360k, 490k\}$ arbitrary square units, $T \in \{0.25, \ldots, 4\}$ arbitrary temperature units in 0.25 increments and $N \in \{1, 2, 3, 4, 8, 16, 24, 32\}$ copies.



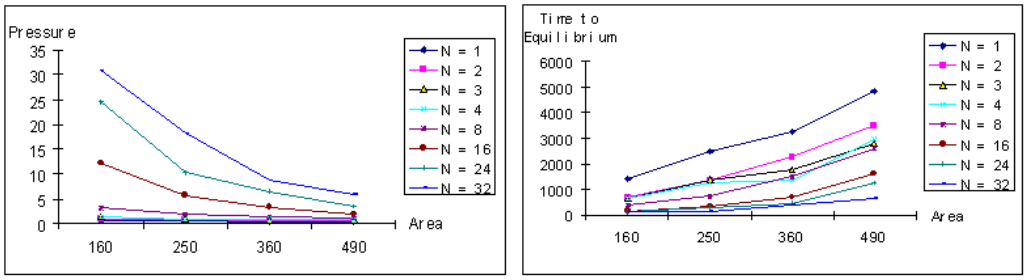
(a) (b)

Fig. 2. Unguided software self-assembly, relationship between (a) pressure and temperature (b) time to equilibrium and temperature (A=360k)

Fig. 2(a) shows the relationship between pressure (P) and temperature (T) as we fixed area of the pool to 360k (600*600) units. As can be seen from Fig. 2(a), regardless of the number of copies of components that are present in the pool, the pressure on the walls of

the pool increase as temperature rises. Figure 2(a) also shows that pressure increases as more copies of components are placed into the pool. In addition, Fig. 2(b) indicates time to equilibrium decreases as temperature increases or number of components increases. This is because when there are more components in a fixed confined space, or if components move faster to explore more areas, it is more likely for them to form valid bindings.

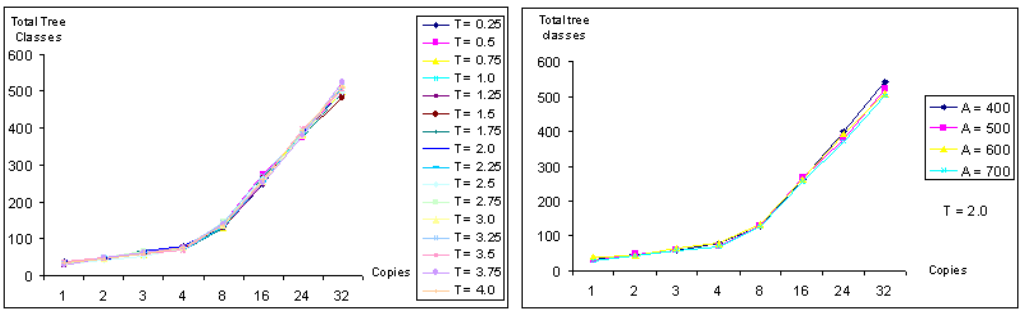
Figure 3(a) shows pressure (P) decreases as area of the pool (A) increases as we fixed the temperature to 2.0. Figure 3(b) illustrates that it takes longer for the system to reach equilibrium as the area of the pool becomes greater. The reason is rather straight-forward, as the pool becomes larger, the probability for a component to bind with another component decreases.



(a) (b)

Fig. 3. Unguided software self-assembly, relationship between (a) pressure and area (b) time to equilibrium and area (T=2.0)

The experimental results from [14] in Fig. 4 illustrate that the number of copies of components placed into the pool is the primary factor that influence the diversity of generated programs. Intuitively, as more components are placed into the pool, self-assembly system yield more results.



(a) (b)

Fig. 4. Using unguided self-assembly: (a) relationship between total different tree classes and number of copies (A = 360k). (b) relationship between total number of different tree classes and number of copies (T = 2.0).

4 Predictive model for $ASAP^2$ with unguided dynamics

The first contribution of this paper is a mathematical model that can be used to predict both t_ε and D_ε for the standard $ASAP^2$ implementation as a function of (A, N, T). With regression software, the following formulae are produced to predict t_ε having one of the three free environment parameters fixed to a pre-specified value (T=2.0, N=8, A=360 respectively). Figure 5 depicts how accurately Eq. 2, 3 and 4 predict t_ε in comparison with the average of the 20 replicas.

$$t_\varepsilon(A, N) = \frac{(7.05 * A) + 321.2}{N} + (3.91 * A) - 593.71 \tag{2}$$

$$t_\varepsilon(A, T) = \frac{(5.21672 * A) - 659.015}{T} + (3.7274 * A) - 416.114 \tag{3}$$

$$t_\varepsilon(T, N) = (\frac{1}{T} + 1) * (\frac{1726}{N} + 637.325) \tag{4}$$

Figure 5 shows that the predictions are fairly accurate as experimental data follow the same trend as predicted by the formulae although errors exist between predictions and experimental results. We quantify the errors by calculating the average and standard deviation of errors rate for each formula (Table 1). Table 1 shows average and standard deviation of errors rate for each equations in different parameter settings. With a lower area setting in Eq. 2 and Eq. 3, the equations predict more accurately with a smaller average error rate. In addition equation 2, 3 predict better than Eq. 4 because they report a smaller average error rate.

$$D_\varepsilon(N) = 15 * N + 16.6 \tag{5}$$

As has been shown in Fig. 4, diversity of self-assembled programs is not greatly affected by temperature neither by area of the pool. Hence, we present Eq. 5 to predict program diversity in relation to number of copies of components only. Figure 5(d) illustrates predicted and the observed experimental results using **unguided** software self-assembly, and the obtained experimental data agree quite well with the model predictions. This can also be seen from Table 1, which shows a small average error rate. This interpolation for (t_ε, D_ε) can already be seen as an advance since similar predictive performance is yet to be achieved with GP.

5 $ASAP^2$ with leaders and dynamic neighbourhood

The second contribution of this paper is a new instantiation of $ASAP^2$ to PSO.

5.1 Model description

We extend here the $ASAP^2$ by introducing *leaders* in the process of software self-assembly. When components are firstly placed into the pool, a pre-specified proportion of components are randomly selected as leaders. If a non-leading component is within a distance threshold

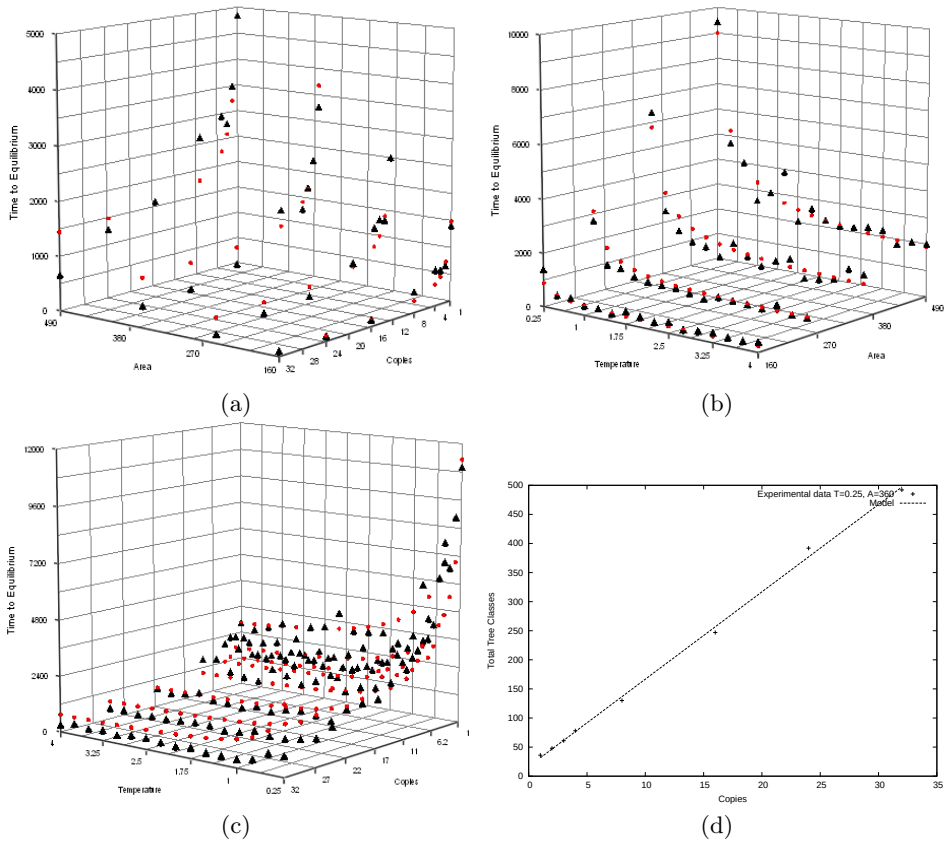


Fig. 5. Prediction model assessment on (a) time to equilibrium with T=2.0 (b) time to equilibrium with N=8 (c) time to equilibrium with A=360k (d) diversity of the generated programs with A=360k, T=0.25. Triangles represent experimental data, and circles represent the corresponding data obtained from our predictive model.

Table 1. Error statistics for predictive model on $ASAP^2$

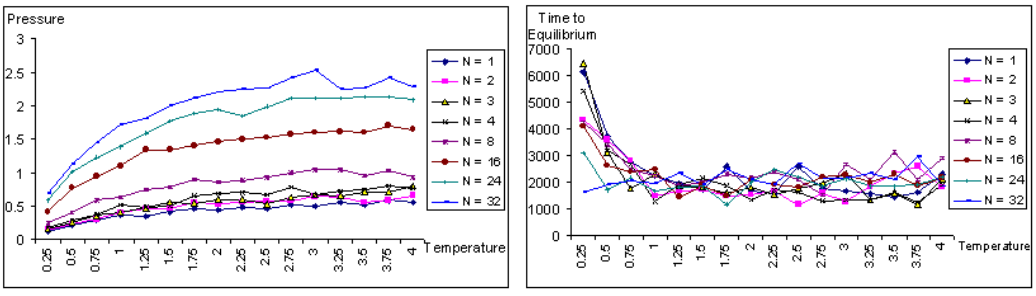
equation	category	average error rate	standard deviation
Eq. 2	A=160	0.208	0.148
	A=250	0.425	0.525
	A=360	0.389	0.473
	A=490	0.261	0.374
Eq. 3	A=160	0.242	0.145
	A=250	0.152	0.146
	A=360	0.192	0.098
	A=490	0.097	0.090
Eq. 4	N=1	0.106	0.106
	N=2	0.129	0.089
	N=3	0.181	0.108
	N=4	0.263	0.095
	N=8	0.224	0.144
	N=16	0.219	0.189
	N=24	0.733	0.207
	N=32	1.190	0.521
Eq. 5	N/A	0.039	0.010

(D_α) to a leader, it will follow the leader component. Leader components or those components that do not have a leader perform a Brownian motion as they do in [14] until a leader appears in its neighbourhood. When a certain component binds with a leader component, the self-assembled structure also becomes a leader. In this way, it can be expected that the proportion of leader components in the pool increases as more components are self-assembled. The motivation behind this change is that the self-assembly process could be made more efficient by introducing leaders since components will attract each other and form groups rather than getting stuck in distant areas of the pool. The idea of our work presented in this paper is inspired by the work done in [15] and [9], which has shown encouraging outcomes on fitness and diversity of the generated results.

If more than one leader is within the threshold range (D_α) of a non-leading component, there needs to be a way to decide which leader to follow. To do that we introduce attractive force of a leader component. Non-leader components follow a leader component that exhibit the greatest attractive force. In this way, a non-leader component changes its leader dynamically in a self-assembly process. The objective is to shorten the time to equilibrium for the system and observe how diversity of the generated populations is affected. Hence, we design the system in such a way that components with more available ports to bind exhibit larger attractive strength. It is important to mention that distance between a leader and a following component contributes to the attractive force too. Hence if a component finds two identical leader components within D_α, it will follow the closer one.

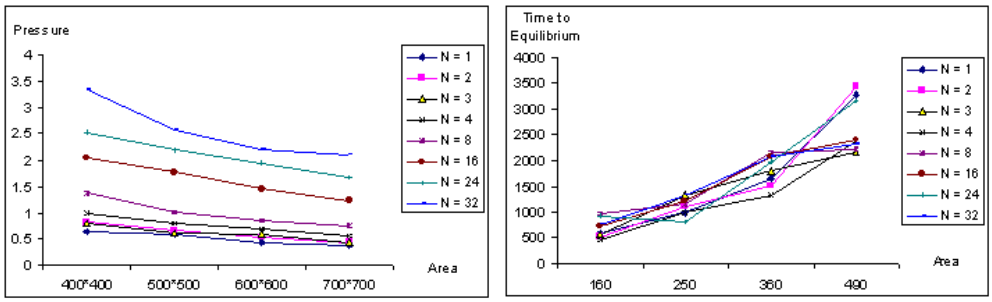
5.2 Experimental results and analysis

The same set of experiments as we did in [14] are performed for our extended system. Figure 6(a) shows the relationship between pressure and temperature, and Fig. 7(a) illustrates the relation between pressure and area of the pool with our extended system. As can be seen, pressure of the environment is affected in a very similar way as the unguided software self-assembly system. That is, pressure increases with more number of components, higher temperature or smaller area of the pool. It can be seen (Fig. 2(a) v.s Fig. 6(a), and Fig.3(a) v.s Fig. 7(a)) that comparing with unguided software self-assembly, the overall pressure on the wall drops. This is because more components are following leaders instead of moving randomly in the pool and hitting the walls.



(a) (b)

Fig. 6. Software self-assembly with leaders: (a) relationship between pressure and temperature with A=360k; (b) relationship between time to equilibrium and temperature with A=360k

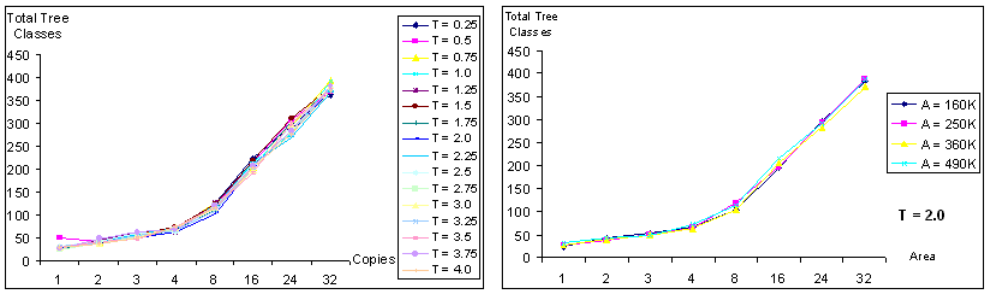


(a) (b)

Fig. 7. Software self-assembly with leaders: (a) relationship between pressure and area with T=2.0; (b) relationship between time to equilibrium and area with T=2.0

Although pressure on the wall is affected by the environment parameters used $ASAP^2$ in very similar ways, time to equilibrium is influenced by those factors in utterly different ways. Figure 6(b) and 7(b) show how time to equilibrium is affected by temperature and area of the pool with various number of copies respectively. It can be seen that the system behaviour has been altered such that number of copies do not play an important role in time to equilibrium. Furthermore, 6(b) indicates that temperature only affects time to equilibrium at a low range (from 0.25 to approximately 1.0). Nevertheless, Fig. 7(b) suggests that area of the pool has a similar influence on time to equilibrium as it does for unguided software self-assembly, i.e. it takes longer for the system to reach equilibrium with a larger pool. Experimental results (Fig. 2(a) v.s Fig. 6(a), and Fig.3(a) v.s Fig. 7(a)) show that time to equilibrium has been significantly reduced when leaders are introduced into the system.

As can be seen from Fig. 8, diversity of the generated population is influenced primarily by number of copies of components. Software self-assembly system with leaders generate fewer parse tree classes than unguided software self-assembly(Fig. 4 and 8).



(a) (b)

Fig. 8. Software self-assembly with leaders: (a) relationship between total tree classes and number of copies in different temperature settings (b) relationship between total tree classes and number of copies in different area settings

To prove that our unguided $ASAP^2$ and extended $ASAP^2$exhibits different behaviour in terms of time to equilibrium and diversity of the generated programs, we performed t-

tests on the experimental data obtained from the two models. Table 2 shows (for A=360), all p-value generated from t-test are smaller than 0.05. By convention, this is considered a strong evidence that the results are statistically different.

Table 2. t-test performed for $ASAP^2$ v.s. extended $ASAP^2$ with area set to 360 for both models

	time to equilibrium	diversity
N=1	$2.61 * 10^{-5}$	0.0088
N=2	0.0044	$1.76 * 10^{-5}$
N=3	0.0097	0.0017
N=4	0.0086	0.0003
N=8	0.0191	$3.61 * 10^{-5}$
N=16	$4.71 * 10^{-9}$	$3.30 * 10^{-11}$
N=24	$9.33 * 10^{-11}$	$2.54 * 10^{-13}$
N=32	$9.06 * 10^{-11}$	$1.47 * 10^{-15}$

6 Assessment of predictive model on extended $ASAP^2$

We examine Eq. 2, Eq. 3, Eq. 4 and Eq. 5 as a predictive model for experimental data within the swarm intelligence inspired system. As can be seen from Fig. 10, greater errors occur between the average of the 20 replicas and predicted outcomes. This can also be seen from Table 3, where the average error rate figures become greater compared with Table 1. This again proves that the behaviour of the system has been altered when leaders are introduced.

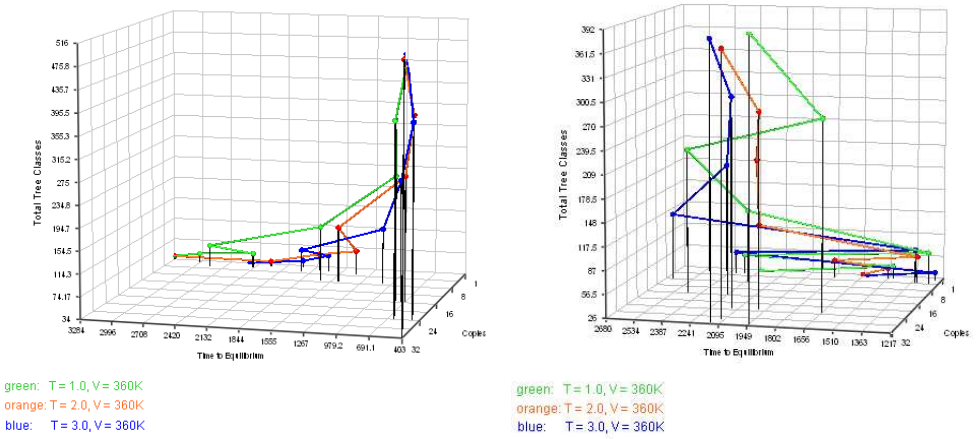


Fig. 9. 3-d chart indicating the inter-relation between total tree classes, time to equilibrium and copies of components in different temperature settings using (a) unguided self-assembly (b) dynamic leaders and neighbourhood

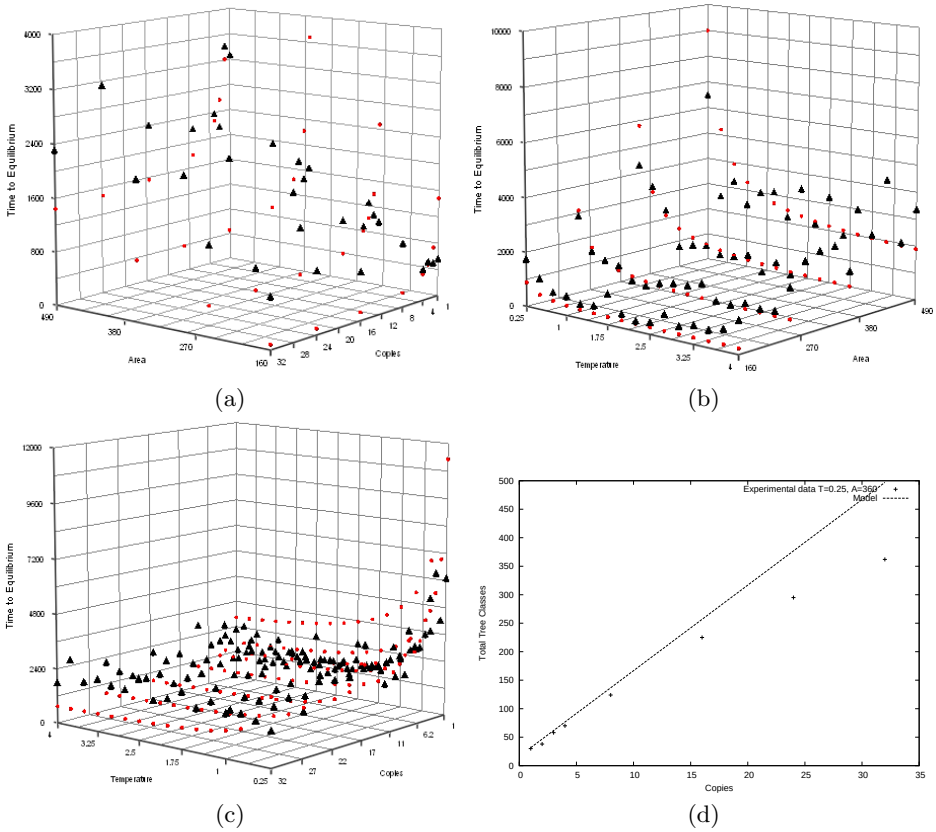


Fig. 10. Prediction model assessment using extended $ASAP^2$ on (a) time to equilibrium with T=2.0 (b) time to equilibrium with N=8 (c) time to equilibrium with A=360k (d) diversity of the generated programs with A=360k, T=0.25. Triangles represent experimental data, and circles represent the corresponding data obtained from our predictive model.

Table 3. Error statistics for predictive model on extended $ASAP^2$

equation	category	average error rate	standard deviation
Eq. 2	A=160	0.706	0.475
	A=250	0.518	0.445
	A=360	0.493	0.354
	A=490	0.289	0.197
Eq. 3	A=160	0.568	0.135
	A=250	0.196	0.154
	A=360	0.209	0.166
	A=490	0.275	0.247
Eq. 4	N=1	0.824	0.299
	N=2	0.363	0.247
	N=3	0.161	0.134
	N=4	0.157	0.136
	N=8	0.442	0.161
	N=16	0.462	0.150
	N=24	0.440	0.208
	N=32	0.561	0.183
Eq. 5	N/A	0.167	0.115

As diversity is mainly affected by number of copies in both systems, we present three dimensional charts to show how it can influence the diversity along with time to equilibrium at the same time. Figure 9 are three dimensional charts to show its tendency, each of which is under a same pre-specified environment setting. Software self-assembly with leader reaches equilibrium much quicker but produces a less diverse population.

7 Conclusion

In our previous work on $ASAP^2$, program gas is used as a metaphor for software components that are placed into a prespecified area container. In this paper, we have reported and contributed the followings. First of all, we have presented a predictive model to interpolate software self-assembly efficiency and diversity for a given set of environment parameter (A, T, N). We can predict the time needed for $ASAP^2$to reach its equilibrium and the resulting diversity of the generated populations.

Secondly, we have also introduced in this paper an extended work on $ASAP^2$. Our work is based on the implementation of "program gas" and ideas from particle swarm optimization methods, in which leaders and dynamic neighbourhood are introduced. A certain proportion of components are selected as leader components, and other components will follow the leaders if they are close within a certain distance. We investigated how diversity of the population and software self-assembly efficiency is affected under the particle swarm regime under the same set of environment parameters (A, T, N). We report the experiments conducted on the extended system, which have shown a significant improvement in software self-assembly efficiency; however, with a slight drop in diversity of the generated population.

Finally, we have assessed our prediction model based on the extended $ASAP^2$. Results have shown greater prediction errors as behaviour of the system has been altered when leaders are introduced. Hence a new set of formulae needs to be derived for the extended system for more accurate predictions. We will report the new equations in our next publication.

Although our extended system may greatly improve the performance of the self-assembly process it is unlikely that this alone will be able to overcome the combinatorial nature of programs space exploration. Instead, more "intelligent" self-assembly should be investigated. Possible solutions may be based on graph grammars for directed self-assembly [13], [12] or the use of a tabu list [7] to ban the bindings between certain components if the binding product is unlikely to be of use.

References

1. Russell J. Abbott. Object-oriented genetic programming, an initial implementation. In *Proceedings of the Sixth International Conference on Computational Intelligence and Natural Computing*, Embassy Suites Hotel and Conference Center, Cary, North Carolina USA, September 26-30 2003.
2. Myriam Abramson and Lawrence Hunter. Classification using cultural co-evolution and genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 249–254, Stanford University, CA, USA, 28–31 July 1996. MIT Press.
3. Giovanni Adorni, Stefano Cagnoni, and Monica Mordonini. Genetic programming of a goalkeeper control strategy for the robocup middle size competition. In Riccardo Poli, Peter Nordin,

William B. Langdon, and Terence C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 109–119, Goteborg, Sweden, 26-27 May 1999. Springer-Verlag.

4. E. Burke, S. Gustafson, G. Kendall, and N. Krasnogor. Advance population diversity measures in genetic programming. *Proceedings of Parallel Problem Solving From Nature*, 2002.
5. Carlos A. Coello and Maximino Salazar Lechuga. A proposal for multiple objective particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC'2002)*, pages 1051–1056, Piscataway, New Jersey, 2002.
6. R.C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, Nagoya, Japan, IEEE Service Center, Piscataway, NJ, 1995.
7. F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.
8. Tad Hogg. Robust self-assembly using highly designable structures. *Papers from the Sixth Foresight Conference on Nanotechnology*, 1999.
9. Eberhart R. C. Hu, X. Multi-objective optimization using dynamic neighbourhood particle swarm optimization. In *Proceeding of the 2002 Congress on Evolutionary Computation*, Honolulu, Hawaii, USA, 2002.
10. G.Hornby J.Lohn and D.Linden. An evolved antenna for deployment on nasa's space technology 5 mission. *In U.-M. O-Reilly et al., editors, Genetic Programming Theory and Practice II, chapter 18. Kluwer, Ann Arbor*, 13-15 May 2004.
11. M.A.Keane J.R.Koza, L.W. Jones and M.J.Streeter. Towards industrial strength automated design of analog electrical circuits by means of genetic programming. *In U.-M. O'Reilly et al., editors, Genetic Programming Theory and Practice II, chapter 8. Kluwer, Ann Arbor*, 13-15 May 2004.
12. Eric Klavins. Directed self-assembly using graph grammars. In *Foundations of Nanoscience: Self Assembled Architectures and Devices*, Snowbird, UT, 2004. Invited Paper. This online version has some corrections.
13. Eric Klavins, Robert Ghrist, and David Lipsky. Graph grammars for self-assembling robotic systems. In *Proceedings of the International Conference on Robotics and Automation*, 2004.
14. L. Li, N. Krasnogor, and J. Garibaldi. Automated self-assembly programming paradigm: Initial investigations. In *Proceedings of the Third IEEE International Workshop on Engineering of Autonomic & Autonomous Systems*, pages 25–34, Potsdamn, Germany, 2006. IEEE Computer Society.
15. Liew K. M. Ray, T. A swarm metaphor for multi-objective design optimization. *Engineering Optimization. 34(2)*, pages 141–153, 2002.
16. Lance Davidson Selvin George, David Evans. A biologically inspired programming model for self-healing systems. In *Workshop on Self-Healing Systems, Proceedings of the first workshop on Self-Healing systems*, pages 102–104. ACM Press, 2002.
17. M. A. Floriano Seung-Yeon Kim, A. Z. Panagiotopoulos. Ternary oil-water-amphiphile systems: Self-assembly and phase equilibria. *Molecular Physics 100*, pages 2213–2220, 2002.
18. Berok Khoshnevis Wei-Min shen, Peter Will. Self-assembly in space via self-reconfigurable robots. *ICRA 2003*, pages 2516–2521, 2003.
19. G.M. Whitesides and B. Grzybowski. Self-assembly at all scales. *Science, v. 295*, pages 2418–2421, 2002.

Evolutionary Algorithms for the Level Strip Packing Problem

Carolina Salto[1], Enrique Alba[2], and Juan M. Molina[2]

[1] Universidad Nacional de La Pampa, La Pampa, Argentina
saltoc@ing.unlpam.edu.ar,
[2] Universidad de Málaga, E.T.S.I. Informática, Universidad de Málaga
Málaga, España
{eat,jmmb}@lcc.uma.es

Abstract. *In this paper, sequential and distributed genetic algorithms are applied to solve the three-stage two-dimensional rectangular strip packing problem. We study problem-specific operators with the aim of improving the quality of final packing patterns. A new operator, denoted as adjustment, is applied after recombination and mutation to all the solutions in the population to improve the filling rate of all levels. The results indicate that the incorporation of the adjustment operator is clearly beneficial for the evolutionary process. We even report that our distributed algorithm is an efficient tool for solving this problem, although the sequential algorithm performs very efficiently from a numerical point of view.*

1 Introduction

The two-dimensional strip packing problem (2SPP) is present in many real-world applications such as in the paper and textile industries, with different constraints and objectives. Typically, the 2SPP consists of a set of M rectangular pieces π_i, characterized by a width w_i and a height h_i, and a larger rectangle with a fixed width W and an unlimited length, designated as the *strip*. The pieces have to be packed with their edges parallel to the edges of the strip (orthogonal cuts). The search is for a layout of all the pieces in the strip that minimizes the required strip length and, where necessary, takes additional constraints into account.

Additionally, another constraint is included in the problem: n-staged guillotine cuts can also be imposed in some cases. This means that the strip has to be cut by stages in which is only possible to perform either horizontal or vertical cuts, but not both. These conditions limit the nesting of horizontal and vertical cuts. In this work, we focus on three-staged guillotineable cutting patterns. This patterns are cut in the following way: in the first stage, horizontal cuts (parallel to W-edge of the strip) are performed producing an arbitrary number of stripes or *levels*. In the second stage, those levels are processed by vertical cuts generating an arbitrary number of so-called *stacks*. The third stage produces the final elements (and waste) from the stacks performing only horizontal cuts in them.

The 2SPP is NP-hard (see the work of Hopper and Turton [8]). A few exact approaches for this problem are known [6, 13]. Regarding the existing surveys of meta-heuristics in the literature, Hopper and Turton [7, 8] reviews the approaches developed to solve 2D packing problems using GAs, but simulated annealing, tabu search, and artificial neural networks are also considered. They conclude that evolutionary algorithms (EAs) [3,14] are the most widely

investigated meta-heuristics in the area of cutting and packing. Lodi et. al [12] consider several methods for the 2SPP in their survey and discussed also mathematical models; specially the case where the items have to be packed into rows forming levels are discussed in detail. Other representative works about GAs applied to solve guillotineable packing problems are [9], [10], [15], and [4].

In this article we use EAs, in particular GAs, as the general driving force to locate the region in which a solution of minimum length is located. The operators used in them allow for final tuning once the solution region is located in the search space. We stress here the inclusion of parallelism in the algorithms to design (hopefully) improved algorithms which reduce the effort to locate a solution. In particular, we analyze the relative advantages of using a single pool of solutions versus having multiple pools of solutions, in which multiple populations evolve independently to solve the same problem with sparse exchanges of information among them (distributed GAs).

In our approach the GA is combined with a heuristic placement routine: the GA is charged for finding the sequence in which the pieces are to be packed, and a second algorithm determines the layout of the pieces onto the strip, following a placement heuristics which respects the three-stage guillotine cutting restrictions. Moreover, a recombination operator and four different mutation strategies are analyzed. In order to reduce the trim loss in each level, an additional final operation is always applied to each generated child. Our contribution is to define a methodology to enhance the basic methods for solving problems larger than the ones found in the literature at present, and to quantify the effects of including specialized operations into the algorithms. We also address a wide analysis of sequential versus distributed algorithms to improve in our understanding on their relative performance.

The paper is organized as follows. A description of some heuristics for level packing is given in Section 2. The components of our evolutionary approach are explained in Sections 3 and 4. In Section 5 we discuss the parameter settings of both the sequential and the distributed versions used in the experimentation. Section 6 reports on the algorithm performances, and finally, in Section 7, we give some conclusions and analyze future search directions.

2 Heuristics for Level Packing

Three classical strategies for the level packing have been derived from famous algorithms for the one-dimensional case [5]. In each case, the pieces are initially sorted by non-decreasing height and packed in the corresponding sequence. Let i denote the current piece, and l the last created level [11]:

- *Next-Fit Decreasing Height* (NFDH) strategy: piece i is packed left justified on level l, if it fits. Otherwise, a new level ($l = l + 1$) is created, and i is placed left justified into it.
- *First-Fit Decreasing Height* (FFDH) strategy: piece i is packed left justified on the first level where it fits, if any. If no level can accommodate i, a new level is initialized as in NFDH.
- *Best-Fit Decreasing Height* (BFDH) strategy: piece i is packed left justified on that level, among those where it fits, for which the unused horizonal space is minimum. If no level can accommodate i, a new level is initialized.

The principal difference among these heuristics lies in the way to select a level to place the piece. NFDH only uses the current level and the levels are considered in a greedy way, meanwhile both FFDH or BFDH analyzes all built levels to accommodate the new piece.

In this work, modified FFDH and BFDH versions are incorporated in the mechanism of a special genetic operator in order to improve the piece distribution. Moreover a modified NFDH version (MNF) is used to build the packing pattern corresponding to a possible solution to the three-stage 2SPP, like the one used in [16] and [17]. This heuristics gets a sequence of pieces as its input, and the cutting pattern is constructed placing pieces into stacks, and stacks into levels in a greedy way, i.e., once a new stack or a new level is started, previous ones are never reconsidered. Specifically, the procedure starts with the first piece π_1 of a permutation $\pi = (\pi_1, \pi_2, ..., \pi_M)$, where M is the number of pieces, and the first level l of the strip is initialized with length $l.length = lenght(\pi_1)$, and the first stack s inside that level with width $s.width = width(\pi_1)$. Once a level is initialized, the next piece, π_i, starts the second stack of the current level if the following conditions are met: $length(\pi_i)$ does not exceed the length of the level, and $width(\pi_i)$ does not exceed the current remaining width of the level, $l.remWidth$. If the following pieces have width equal to $width(\pi_i)$ they are stacked one under the other as long as the length of the stack, $s.length$, does not exceed $l.length$. If the width of the next piece π_j is different from $width(\pi_i)$ or $l.length$ is exceeded by $s.lenght$ then a new stack is started with π_j if $widht(\pi_j) \leq l.remWidth$, else a new level is started with π_j. This process is repeated until no pieces remain in π. This heuristics also allows stacked elements to extend a level's length if the total waste of the level decreases.

3 The Genetic Approach

By simulating evolution, a GA maintains a population of multiple individuals which evolve throughout multiple generations by allowing the reproduction and further enhancement of the fittest ones.

The fact that GAs work with a population of individuals of considerable length for a complex problem usually means a significant utilization of computational resources. A clear way to decrease these requirements is to resort to parallel families of GAs. Parallel GAs modify the typical behavior of the equivalent sequential algorithm since they actually are using a structured population. Among the most widely known types of structured GAs, *distributed* (dGA) and *celular* (cGA) algorithms are very popular optimization procedures [1]. In short, separating one single population (panmixia) into several subpopulations (decentralization) is a healthy line of research since not only the resulting techniques are faster in terms of numerical cost (visited points in the problem landscape), but they also can be easily parallelized to perform a larger number of steps per time unit in a cluster of workstations or any other multiprocessor system.

In a dGA, there exist many elementary subalgorithms (island GAs or demes) working on separate populations. All these subalgorithms are intended to perform the same reproductive plan and communicate at a given frequency by exchanging individuals, by following a particular communication pattern. A migration policy controls the kind of distributed GA being used. It must define the island topology, when migration occurs, which individuals are being exchanged, the synchronization among the sub-populations, and the kind of integration of exchanged individuals within the target subpopulations. Therefore, dGAs are interesting not just because they are a faster version of sequential GAs, but also because they provide a different search mechanism, frequently more efficient.

In most cases, applying a GA means devising a representation for encoding a candidate solution. We encode a cutting pattern in a chromosome as a sequence of pieces that defines the input for the layout algorithm (MNF), and we use a natural number to name each piece. Therefore, a chromosome will be a permutation $\pi = (\pi_1, \pi_2, ..., \pi_M)$ of M natural numbers.

GAs are guided by the values computed by an objective function for each tentative solution until the optimum or an acceptable solution is found. In our problem, the objective is to minimize the strip length (*strip.length*) needed to build the layout corresponding to a given solution π. But two packing patterns can have the same length —so their fitness values will be equal— although, from the point of view of reusing the trim loss, one of them can be actually better because the trim loss in the last level (which still connects with the remainder of the strip) is greater than the one present in last level in the other layout. Therefore we are using the following, more available, fitness function:

$$F(\pi) = strip.length - \frac{1}{l.waste} \tag{1}$$

where *strip.length* is the length of the packing pattern corresponding to the permutation π and $l.waste$ is the area of reusable trim loss in the last level of the packing pattern.

4 Genetic Operators for solving 2SPP

In this section we will describe three groups of operators, based on their actions and results (recombination, mutation, and adjustment), which were applied in all our algorithms. The idea behind the operators is to change the location of the pieces so that the final cost is reduced. Most operators work in terms of the filling rates of the levels. This rate is calculated as follows for a given level l:

$$fr(l) = \sum_{i=1}^{n} \frac{width(\pi_i) \times length(\pi_i)}{W \times l.length} \tag{2}$$

where n is the number of pieces in l, $width(\pi_i)$ and $length(\pi_i)$ are the dimensions of such pieces, and W and $l.length$ the level dimensions.

The recombination operator, named as *best inherited levels recombination* (BIL), transmits the best levels to the child, i.e. those with the biggest filling rate or, equivalently, with the least trim loss. In this way, the inherited levels can be kept or readjusted if one of them takes some pieces from the following level. This depends on how compacts the levels are: if they are very compact the levels should possibly be kept up and do not take pieces out of its neighboring level. The actual recombination works as follows. In the first step the filling rate of all levels from one parent, $parent_1$, is calculated. After that, a selection probability for each level l proportional to the filling rate is determined. A number k of levels are selected from $parent_1$ regarding proportional selection according to the filling rate. The pieces π_i belonging to the inherited levels are placed in the first positions of the child. Meanwhile, the remaining positions are filled with the pieces which do not belong to that levels in the order they appear in the other parent $parent_2$. Figure 1 gives an example for the level transfer in the course of a recombination operation and also the filling process of the remaining positions.

The mutation operators we have considered are the following ones:

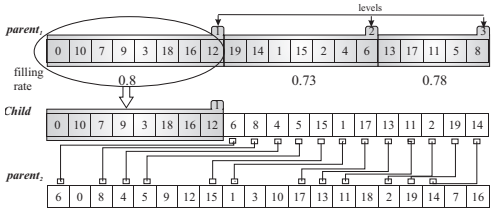


Fig. 1. Example of BIL recombination.

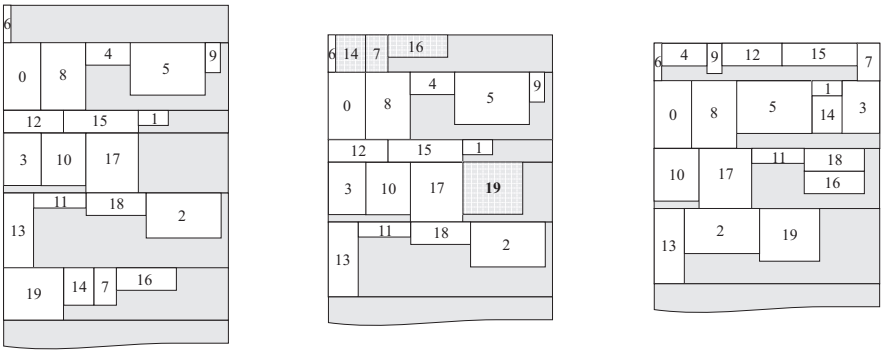


Fig. 2. Example of layouts obtained by applying some operators. Original layout (left). The layout after applying the LLR mutation (middle) and the MFF_Adj operator (right).

1. *Piece Exchange (PE):* this operator selects two genes (pieces) randomly in a chromosome and exchanges their positions.
2. *Stripe Exchange (SE):* this operator selects two levels of the cutting pattern, at random, and exchanges (swaps) these levels in the chromosome.
3. *Best and Worst Stripe Exchange (BW_SE):* in this mutation, the best and the worst level are always moved. The best level (the one with highest filling rate) are relocated at the beginning of the chromosome while the worst level are moved to the end.
4. *Last Level Rearrange (LLR):* This operator takes the first piece π_i of the last level, and analyze all levels in the layout —following the MNF heuristics— trying to find a place for that piece. If this search is not successful the piece π_i is not moved at all. This process is repeated for all of the pieces belonging to the last level. During this process, the piece positions inside the chromosome are consistently rearranged. Figure 2 introduces an example (middle layout), taken as reference the left layout; in this layout, shadow pieces belonged to the last level of the original layout, which has disappeared after the mutation reducing the total strip length.

Regarding SE and BW_SE, the movements can help to the involved levels or its neighbors to take pieces from neighboring levels improving their trim loss.

Finally, we proceed to describe the new operator devised. This operator is applied after recombination and mutation to all the solutions in the population with the aim of improving the filling rate of all levels. Given a solution, the operator consists of the application of a modified version of FFDH (*MFF_Adj*) or BFDH (*MBF_Adj*) heuristics. The possible new layout obtained in this way has to be transmitted to the chromosome in such a way that we can obtain the same layout by applying MNF to the chromosome.

MFF_Adj works as follows. It considers the pieces in the order given by the permutation π. The piece π_i is packed into the first stack in the first level it fits, as in MNF. If piece π_i does no fit into any existing stack and there is room enough in that level, a new stack is created, as in MNF. Otherwise the following levels are considered and checked in the previous order. If no space were found, a new stack containing π_i is created and packed into a new level in the remaining length of the strip. The above process is repeated until no piece remains in π. An example is shown in Figure 2 (right). In this case the corresponding chromosome has to be rearranged.

MBF_Adj proceeds in a similar way, but with the difference that a piece π_i is packed on the stack (among those that can accommodate it) for which maximizes the filling rate of the corresponding level. If no place is found to accommodate the piece, a new stack is created containing π_i and packed into a new level, as in MNF. These steps are repeated until no piece remains in the permutation π and, finally, the resulting layout yields a suitable reorganization of the chromosome.

5 Implementation

As we said at the beginning, sequential and distributed GAs have been run in order to evaluate their effectiveness. Each GA approach has been tested with different combinations of mutation (PE, SE, BW_SE and LLR) and adjustment (MFF_Adj and MBF_Adj) operators and three instances of the strip packing problem with 100, 150 and 200 pieces respectively, with integer dimensions, and a strip with width $W = 100$. These instances have been randomly generated by an implementation of a data set generator following the ideas proposed in [18].

We will compare the results obtained with a sequential steady state GA (GAseq) and two distributed steady state GA versions: one running in a single processor system (GAdist1), and the other running in a cluster of eight workstations (GAdist8). These algorithms were run with MALLBA [2], a C++ software library fostering rapid prototyping of hybrid and parallel algorithms. All the algorithms were run until a maximum number of evaluations was reached. First, we will contrast GAseq versions with and without the application of the adjustment operator, to show its numerical advantages, and then we will compare the GAdist versions.

The steady state GA uses a binary tournament selection for each parent, and the new generated individual replace the worst individual in the population only if it is better. The population size for GAseq is set to 512 individuals. The initial population is randomly generated. The maximum number of evaluations is 65,536. The recombination operator is BIL with an application probability of 0.8, while the mutation probability was set to 0.1. After these genetic operators, the adjustment operator is applied to all offspring with a probability equal to 1.0. The parallel distributed island model has 512/8 individuals per island. The migration will occur in a unidirectional ring manner every 512 evaluations with asynchronous communications for efficiency, sending one single randomly chosen individual to the neighbor subpopulation. The target population incorporates this individual only if it is better than its presently worst solution. The above parameters have been previously tuned, but we do not include the tuning details in the article due to room restrictions. We have used a cluster composed of 16 machines with the following characteristics: Pentium 4 at 2.4 GHz and 512 MB RAM. The operating system used is SuSE Linux with 2.4.19-4GB kernel version. See Table 1 for a summary of the parameters.

Table 1. Parameters used for the different GAs

Parameter	GAseq	GAdist1	GAdist8
amount of processors	1	1	8
amount of processes	1	8	8
population size	512	64	64
recombination	BIL		
mutation	PE, SE, BW_SE and LLR		
adjustment operator	MFF_Adj and MBF_Adj		
replacement strategy	$(\mu + 1)$		
stop criterion	(65,536 evaluations)		
interconnection topology	-	unidirectional ring	
migration rate	-	every 512 evaluations	
number of individuals to migrate	-	1	
methods to select migrants	-	uniform random	
policy to replace with immigrants	-	replace worst if migrant is better	

Table 2. Experimental results for GAseq for all instances.

Inst	**Mut_op**	**GAseq**			**GAseq+MFF_Adj**			**GAseq+MBF_Adj**		
		best	***avg*** $\pm\sigma$	$eval_{best}$	***best***	***avg*** $\pm\sigma$	$eval_{best}$	***best***	***avg*** $\pm\sigma$	$eval_{best}$
100	PE	237	247.07 ± 3.95	26928.90	226	231.03 ± 1.97	9485.50	228	233.97 ± 3.01	11299.90
	SE	235	246.09 ± 4.54	26842.53	226	231.30 ± 2.38	13331.33	225	232.83 ± 3.53	11354.57
	BW_SE	237	251.13 ± 7.03	6773.97	227	231.00 ± 2.38	3780.13	229	233.27 ± 2.59	3682.40
	LLR	240	250.37 ± 5.46	6475.67	226	232.10 ± 2.34	3675.33	229	233.80 ± 2.47	3565.93
150	PE	242	254.27 ± 5.94	16485.87	228	233.13 ± 2.98	18829.97	225	236.90 ± 4.94	15801.63
	SE	239	252.73 ± 6.31	33231.67	229	233.70 ± 3.10	12573.37	224	235.87 ± 5.11	12028.33
	BW_SE	243	255.83 ± 5.63	7602.07	228	233.24 ± 2.82	5676.55	229	235.67 ± 4.20	5863.33
	LLR	243	255.33 ± 5.41	7788.60	225	233.31 ± 4.10	5676.34	232	238.30 ± 3.37	5565.17
200	PE	248	256.13 ± 4.84	19900.33	226	230.73 ± 2.94	11381.33	223	230.87 ± 3.70	8908.40
	SE	247	254.49 ± 4.56	44749.90	223	230.67 ± 3.21	8404.20	224	233.13 ± 3.84	12549.07
	BW_SE	248	258.03 ± 5.69	8949.13	222	231.10 ± 3.58	4901.67	221	230.86 ± 4.02	5094.97
	LLR	248	258.43 ± 5.64	8110.93	225	231.43 ± 3.32	5586.40	222	231.23 ± 4.06	4305.27
mean		242.25	253.32	17819.96	225.92	231.89	8608.51	225.92	233.89	8334.91

6 Computational Analysis

In this section we will summarize the results of using the three proposed algorithms with its variants (mutation and adjustment operators) on all the problem instances. For each algorithm we have performed 30 independent runs per instance using the parameter values written in the previous section. Our aim is to offer meaningful results from a statistical point of view. Also we computed a Student t-test analysis so that we could be able to distinguish meaningful differences in the average values. The significance p-value is assumed to be 0.05, in order to indicate a 95% confidence level in the results.

Let us first begin with the results of the computational experiments for the sequential variants in order to justify the application of the adjustment operator. Average results over all instances obtained by sequential algorithm variants are listed in Table 2. The most relevant aspects that were measured in this comparison are the following ones: the best fitness value obtained (column *best*), the average objective values of the best found feasible solutions along their standard deviations (column *avg*), and the average number of evaluations needed to reach the best value (numerical effort) (column $eval_{best}$). The last row in Table 2 shows average results over all the instances obtained by each algorithm just as a summary of the trends. The minimum *best* values are printed in bold.

We must notice that the GAseq variants applying some of the adjustment operators (MFF_Adj or MBF_Adj) outperformed significantly the plain GAseq in both aspects: solution quality and number of evaluations (p-values well below 0.05). To confirm this results,

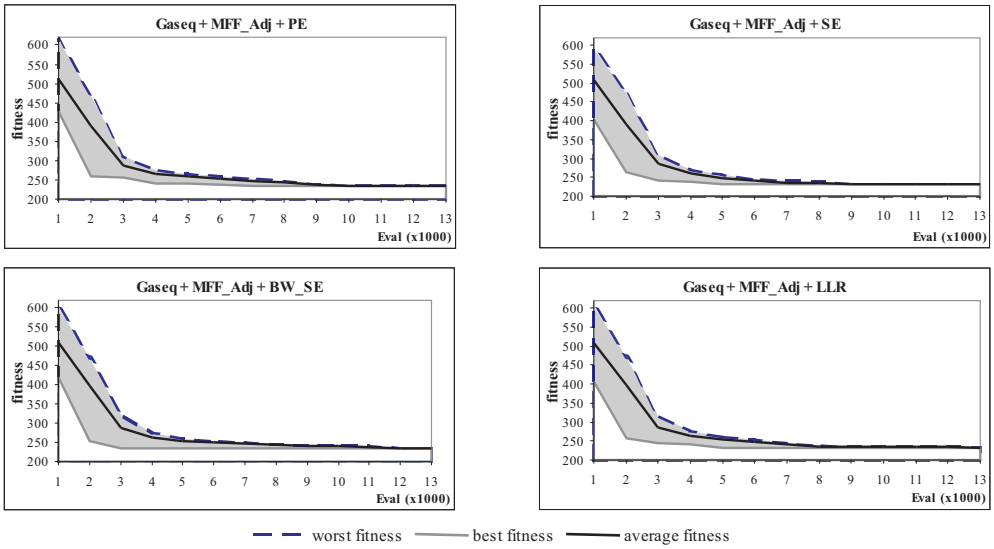


Fig. 3. Illustrating evolution for problem instance 150.

we used the test of multiple comparisons of Bonferroni. The test verified the differences among the results of each group (plain GAseq versus GAseq applying some of the adjustment operators), but remarking that there not differences inside each group. The reason of the outperforming is based on the improvement in the pieces layout obtained after the application of these operators, where its goal is to rearrange the pieces in order to reduce the trim loss inside each level. However, it can be observed that the average best solution quality does in general not differ very much between algorithms applying the two adjustment operators (the mean best in last row are quite similar) and differences are not significant, since the t-test offers a p-value clearly larger than 0.05. Besides, regarding the average objective values of the best found feasible solutions, GAseq with MFF_Adj was more stable than the algorithm with MBF_Adj, producing best average values and also small deviations. As to the influence of mutation in the final solution quality, it turns out that no mutation operator is consistently best for all tested instances. On average, however, a little advantage is observed in favor to BW_SE and LLR operators (also corroborated by the Bonferroni test). Moreover, these operators are more robust since they employed half the number of evaluations that PE and SE to reach their best values. The reason is that BW_SE and LLR mutations should produce more variations in the layout of the pieces helping levels to take pieces from neighboring levels finally improving their trim loss.

In order to illustrate how the best, mean, and worst fitness values evolve, Figure 3 presents an example for instance 150 taking the GAseq using MFF_Adj operator. Similar charts are obtained with other instances and also under MBF_Adj operator. The gray area in Figure 3 represents the different solutions in the population at a given moment. It can be observed that all the algorithms showed a pronounced improvement in fitness values during the first 2000 evaluations; after that, the progress continued but at a low pace until near evaluation 10000, where the algorithms start stagnation.

Let us now discuss the process of solving the 2SPP by the GAdist algorithms. In this case the algorithms are tested always using adjustment operators due to the improvement

Table 3. Experimental results for GAdist1/GAdist8 with MFF_Adj.

Inst	Mut_op	GAdist1					GAdist8				
		best	***avg*** $\pm\sigma$	$eval_{best}$	t_{best}	t_{total}	***best***	***avg*** $\pm\sigma$	$eval_{best}$	t_{best}	t_{total}
100	PE	227	231.70 $\pm$2.62	22103.73	963.19	2835.40	**224**	231.70 $\pm$3.40	11747.90	73.27	414.73
	SE	**224**	231.66 $\pm$3.06	22121.43	980.64	2850.16	227	232.00 $\pm$3.16	13500.20	88.16	388.79
	BW_SE	**226**	234.00 $\pm$3.30	632.93	21.11	2943.85	228	235.00 $\pm$4.60	303.70	0.95	387.59
	LLR	**227**	234.43 $\pm$3.88	345.63	7.25	2883.00	231	235.20 $\pm$3.01	356.60	1.36	363.45
150	PE	**223**	236.13 $\pm$4.26	27438.30	5019.26	11768.40	231	236.50 $\pm$4.20	25561.90	593.54	1555.77
	SE	**229**	235.20 $\pm$3.21	17706.97	3186.71	11529.92	**229**	232.87 $\pm$2.36	22089.38	349.62	1476.81
	BW_SE	232	238.08 $\pm$3.88	576.96	77.20	12154.15	**231**	238.60 $\pm$4.25	894.10	15.15	1486.04
	LLR	**229**	237.69 $\pm$3.65	749.24	105.04	11945.15	232	236.80 $\pm$3.33	592.50	8.91	1583.81
200	PE	**226**	233.54 $\pm$3.35	9487.00	3116.82	22149.75	227	232.80 $\pm$2.46	23298.33	1157.14	3362.66
	SE	**229**	233.67 $\pm$2.87	26684.58	9716.44	23698.58	**229**	233.27 $\pm$3.17	23491.13	1101.30	3028.76
	BW_SE	**227**	232.80 $\pm$3.39	459.30	120.96	25454.23	**227**	232.20 $\pm$3.16	667.10	22.98	3361.62
	LLR	**228**	234.22 $\pm$4.99	427.89	101.59	22163.07	229	234.87 $\pm$2.88	442.07	14.08	2977.36
mean		227.25	234.43	10727.83	1951.35	12697.97	228.66	234.42	10410.72	294.19	1702.57

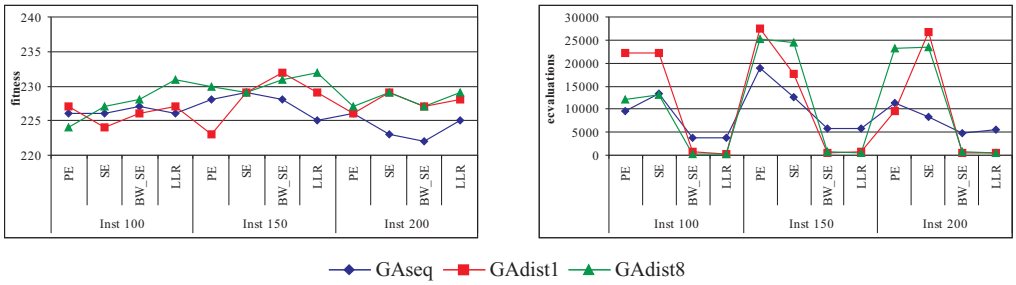


Fig. 4. Comparison of each algorithm variant using MFF_Adj. Best solution obtained by each algorithm (left) and mean number of evaluations to reach the best value(right).

in the results observed in the precedent sequential study. So the analysis is divided into two parts regarding the two adjustment operators.

Table 3 shows the results obtained for the GAdist variants using MFF_Adj operator for each instance and each mutation operator. In addition to *best*, *avg* and $eval_{best}$ results, here we also report the average time spent to find the best value (column t_{best}), and the run time expressed in seconds (column t_{total}).

In this case both GAdist1 and GAdist8 were able to find solutions of the same quality each other. This finding confirms that the two algorithms are performing basically the same search, with only time being a significant difference between them (p-value well below 0.05) since they run in 1 and 8 processors, respectively. Regarding the effects of mutation, here again no mutation operator is consistently the best for all tested instances, and the difference in the number of evaluations in favor to BW_SE and LLR is present once more. (also corroborated by the Bonferroni test). The GAseq improved the average fitness values (see the last summary row with the trend in Tables 2 and 3 or Table 5 for a summary of the trend) but most of them are of the same statistical significance than those found by the distributed ones (see Figure 4 (left) for more details). As to the numerical effort, GAdist1 and GAseq are performing similarly, sometimes one of them outperform the other in number of evaluations (Figure 4 (right)).

Table 4 and Figure 5 present the results obtained for the GAdist variants applying the MBF_Adj operator. Here again we can notice that the results are much the same: the two GAdist versions seem again to provide the same average best solutions and evaluations

Table 4. Experimental Results for GAdist1/GAdist8 with MBF_Adj.

Inst	Mut_op	GAdist1					GAdist8				
		best	***avg*** $\pm\sigma$	$eval_{best}$	t_{best}	t_{total}	***best***	***avg*** $\pm\sigma$	$eval_{best}$	t_{best}	t_{total}
100	PE	**230**	233.58 $\pm$2.43	21273.63	1063.24	3299.91	**230**	234.50 $\pm$2.51	17601.90	130.06	467.76
	SE	**226**	232.94 $\pm$3.51	23497.71	1279.85	3476.85	**226**	233.80 $\pm$3.99	28216.50	214.98	480.92
	BW_SE	**230**	236.50 $\pm$4.14	360.40	11.53	3440.33	232	237.50 $\pm$3.17	307.90	1.27	468.40
	LLR	232	236.10 $\pm$3.18	281.30	6.98	3573.31	**230**	235.22 $\pm$4.55	730.56	4.08	443.21
150	PE	234	239.40 $\pm$3.24	17565.70	3125.03	12011.52	**232**	239.60 $\pm$4.84	16092.30	394.89	1659.36
	SE	233	241.50 $\pm$4.58	17934.60	3392.63	12041.60	**228**	239.70 $\pm$4.92	10416.10	218.42	1578.09
	BW_SE	**233**	241.40 $\pm$3.80	520.85	74.25	13746.30	235	241.00 $\pm$3.37	576.20	10.64	1691.62
	LLR	**233**	241.78 $\pm$4.92	570.79	75.85	12925.06	234	240.80 $\pm$3.88	653.70	13.15	1791.28
200	PE	227	233.70 $\pm$3.47	17113.30	6262.42	24044.05	**223**	233.40 $\pm$4.38	18057.50	781.09	2869.99
	SE	**229**	234.50 $\pm$3.63	21962.20	8512.13	23525.90	**229**	232.70 $\pm$2.67	20453.30	984.61	3332.03
	BW_SE	**229**	235.20 $\pm$2.78	450.10	125.75	27878.52	230	231.66 $\pm$1.37	1025.17	38.19	2895.27
	LLR	233	239.28 $\pm$5.39	530.43	94.80	17076.75	**230**	234.80 $\pm$2.70	476.20	15.72	3538.57
mean		230.75	237.16	10171.75	2002.04	13086.67	229.92	236.22	9550.61	233.92	1768.04

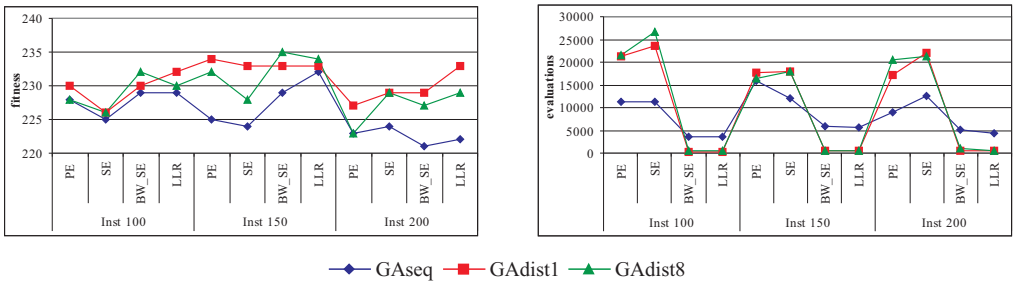


Fig. 5. Comparison of each algorithm variant using MBF_Adj. Best solution obtained by each algorithm (left) and mean number of evaluations to reach the best value (right).

Table 5. Summarize Results for GAseq, GAdist1 and GAdist8.

Alg.	MFF_Adj				MBF_Adj			
	best	***avg***	$eval_{best}$	t_{total}	***best***	***avg***	$eval_{best}$	t_{total}
GAseq	225.92	231.89	8608.51	1554.85	225.92	233.89	8334.91	1616.32
GAdist1	227.25	234.43	10727.83	12697.97	230.75	237.16	10171.75	13086.67
GAdist8	228.66	234.42	10410.72	1702.57	229.92	236.22	9550.61	1768.04

(p-value well below 0.05). Besides, the execution time relationship is similar to the previous operator (p-value larger than 0.05).

From our results, it is clear that the two GAdist algorithms are similar in solution qualities and number of evaluations while GAseq is clearly computing very good average values for these two metrics (see Table 5 with mean values of the trend). Therefore, the conclusions are that, numerically speaking, GAseq is a very competitive algorithm for any of the adjustments operators. The reason for such result is that the distributed search is not really targeted to combine building blocks, and since the evaluations and migrations proceed so fast, our distributed GAs do not have a chance to evolve independently, and instead converge too quickly to global optima. It can also be concluded that there is a little advantage for those algorithms applying MFF_Adj operator if we look at their mean best solution found and also concerning the execution time.

7 Conclusions

In this paper we intend to design better algorithms to solve the 2SPP. We apply sequential and distributed genetic algorithms and also different operators for the search for good packing patterns.

Our results show that the incorporation of the new adjustment operator into the evolutionary process works properly providing a faster sampling of the search space. But regarding the final solution quality, the algorithms do not present significant differences. Additionally, GAseq is better than the GAdist variants in accuracy for the most tested instances; this is because there are many constraints in a very large search space and there is no time for a separate search since the local search operators provoke a too fast convergence in any of the tested algorithms. We postulated that the used problem interpretation is decoupling the search of the algorithms from the actual problem. In the future we will investigate non-permutation representations and a direct mapping to the final disposition of the pieces.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Education and the European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project, http://oplink.lcc.uma.es). We also acknowledge the Universidad Nacional de La Pampa, and the ANPCYT in Argentina from which we received continuous support.

References

1. E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, 2005.
2. E. Alba, J. Luna, L.M. Moreno, C. Pablos, J. Petit, A. Rojas, F. Xhafa, F. Almeida, M.J. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, and C. León. *MALLBA: A Library of Skeletons for Combinatorial Optimisation*, volume 2400 of *LNCS*, pages 927–932. Springer, 2002.
3. T. Bäck, D. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. Oxford University Press, New York, 1997.
4. A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Jouranal of Operational Research (article in press)*, 2005.
5. E.G Coffman, M.R. Garey, D.S. Johnson, and R.E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9:801–826, 1980.
6. S.P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithm. Technical Report ZPR97-290, Mathematisches Institut, Universitt zu Kln, available from the first author at Department of Mathematics, 1997.
7. E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, University of Wales, Cardiff, U.K., 2000.
8. E. Hopper and B. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.
9. S. Hwang, C. Kao, and J. Horng. On solving rectangle bin packing problems using genetic algorithms. *IEEE International Conference on Systems, Man, and Cybernetics - Humans, Information and Technology*, 2:1583–1590, 1994.
10. B. Kroger. Guillotineable bin-packing: a genetic approach. *European Journal of Operational Research*, 84:645–661, 1995.

11. A. Lodi, S. Martello, and M. Monaci. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematicsournal of Operation Research*, 141:241–252, 2002.
12. A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operation Research*, 141:241–252, 2002.
13. S. Martello, S. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *Informs Journal on Computing*, 15:310–319, 2003.
14. M. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, third revised edition, 1996.
15. C. Mumford-Valenzuela, J. Vick, and P. Wang. *Metaheuristics: Computer Decision-Making*, chapter Heuristics for large strip packing problems with guillotine patterns: An empirical study, pages 501–522. Kluwer Academic Publishers BV, 2003.
16. J. Puchinger, G.R. Raidl, and G. Koller. *Solving a Real-World Glass Cutting Problem*, volume 3004 of *LNCS*, pages 162–173. Springer, 2004.
17. C. Salto, J.M. Molina, and E. Alba. Sequential versus distributed evolutionary approaches for the two-dimensional guillotine cutting problem. *Proceedings of International Conference on Industrial Logistics (ICIL 2005)*, pages 291–300, 2005.
18. P. Wang and C. Mumford-Valenzuela. Data set generation for rectangular placement problems. *EJOR*, 134:378–391, 2001.

From Cooperative Team Playing to an Optimization Method

Xiaofei Huang

School of Information Science and Technology
Tsinghua University, Beijing, P.R. China, 100084
huangxiaofei@ieee.org

Abstract. *Team playing is a common cooperative behavior among human beings where the team members working together through cooperation can solve hard problems which are beyond the capability of any member in the team. Some cooperation strategies used in team playing have recently inspired a general optimization method, called cooperative optimization, for attacking difficult optimization problems. It has shown promising results at solving real-world optimization problems. A cooperative optimization algorithm also has a number of computational advantages because it knows whether a solution it found is a global optimum and which direction is more promising than others at finding global optimal solutions. This paper elaborates more on the inspiration behind this optimization method and its power and limitations.*

1 Introduction

Cooperation is an ubiquitous phenomenon in nature. We often times see in nature a swarm of insects, a school of fish, a group of wolfs or lions work together like a team to achieve certain goals, such as running away from their predators or catching their preys. We can view those as cooperative systems where agents in a system work together through some cooperation mechanisms to achieve certain goals. Usually, cooperation can benefit the agents in a cooperative system more than they can receive if they work independently.

Within living organisms we can also see cooperation at many different levels, from proteins, to cells, tissues, and organs. Beyond the scope of living organisms, we often see cooperation at all different levels and scopes within species and cross species.

Human society is probably the largest cooperative system in nature. We work together to achieve many astonishing goals that are beyond the imagination of many of us. Examples are sending people into space, exploring nuclear energy, inventing electronic computing devices that can carry billions of operations per a second, and many more.

Different cooperative systems in nature may have different mechanisms for cooperation. For the same system, there may exist more than one mechanism for cooperation, different mechanisms operated at different levels, and different mechanisms employed at different times or different situations. Furthermore, the mechanisms may also evolve together with the evolutions of cooperative systems.

1.1 Mathematical Formalization of Cooperative Systems

We can discover, abstract, and theorize different cooperative systems and generalize them to solve computational problems for us. The challenge is to build mathematical models to capture the essential aspects of cooperation systems. It implies that the models should enable us to study the

computational properties of cooperative systems instead of other irrelevant properties. The models should also enable us to translate them into working optimization methods.

Different cooperative systems can have different structures for organizing agents in a system and deploy different cooperation mechanisms for defining the interactions among the agents in a system. They could inspire different optimization methods suitable for different purposes. Due to the richness of cooperation systems existed in nature, we can surely model many optimization methods based on them.

There is a special kind of cooperative systems useful at inspiring optimization methods. A system of this kind has the characteristic that its goal can be measured numerically and it depends on multiple variables controlled by the agents in the system. The agents in the system are autonomous and locally interact with their neighbors. Also, no special agent in the system dictates others and no centralized control to manipulate the agents too. Hence, those cooperative systems are decentralized and self-organized.

Often times, a decentralized, self-organized cooperative system can have many interesting emerging behaviors through simple, local interactions of the agents in the system. Different ways of decision makings by the agents in the system and different patterns of local interactions among those agents define different cooperation mechanisms for the system.

The cooperative optimization method proposed before [1, 2] is inspired by one cooperation mechanism in decentralized, self-organized systems. It is a general method for attacking hard optimization problems with impressive performance at solving many real-world applications [3–5] of large scales.

1.2 Existing Metaheuristic Methods

There is a special kind of artificial intelligence technique, called Swarm Intelligence, which is also based on the study of collective behaviors in decentralized, self-organized systems. Therefore, the cooperative optimization method is a form of swarm intelligence. Some of the existing, best-known swarm intelligence techniques are Ant Colony Optimization [6] and Particle Swarm Optimization [7].

Those existing swarm intelligence techniques can also be understood as optimization methods inspired by different cooperation mechanisms found in nature. Those cooperation mechanisms are different from the cooperation mechanism that the cooperative optimization is based on. When we study cooperative systems in nature, we might come out with different understandings about the underlying cooperation mechanisms. This is caused by different points of view we take, the huge variety of cooperative systems existing in nature, and many different levels and scopes of cooperation. Therefore, it can be fruitful for us to discover new optimization methods by studying cooperation mechanisms in nature.

Ant Colony Optimization and Particle Swarm Optimization are classified as metaheuristic methods for optimization. Other metaheuristic methods are simulated annealing [8–11], tabu search [12], guided local search [13], randomized adaptive search procedures [14], iterated local search, evolutionary computation [15–17], scatter search, and stochastic diffusion search. They have been applied to attack a large variety of optimization problems.

Although there are many reported successes of applied metaheuristic methods at solving different optimization problems, there are also criticisms about those methods. Those criticisms pointed out that one couldn't say that any metaheuristic method is better than any other. For instance, one couldn't prove that any metaheuristic method is better than brute-force search or pure random search. To minimize an objective function, for another instance, one cannot prove that a metaheuristic using a hill-descending heuristic is better than the same metaheuristic using a hill-climbing heuristic. The "no free lunch" (NFL) theorems [18] have demonstrated that the average performance for any pair of black box optimization algorithms across all possible problems is identical. The NFL theorems imply that if some algorithm's performance is superior to that of another

algorithm over some set of optimization problems, then the reverse must true over the set of all other optimization problems.

In author's personal view, the problem with metaheuristic optimization methods presented above is caused by the lack of general global optimality conditions for those methods to identify global optimal solutions within a polynomial time. It is also true for other general nonlinear optimization methods (see Chapter 5 in [19] for a comprehensive survey) such as Armijo's line-search algorithm, gradient methods, conjugate gradient methods, Newton's method, Quasi-Newton methods, and derivative-free methods. Without any global optimality condition, an optimization algorithm is just a black-box optimization algorithm. Its performance at finding global optimal solutions is governed by the NFL theorems. It does not know where to find global optimal solutions and whether a solution it found is a global optimum. Because of that, they do not know how to organize their optimization processes effectively and when to terminate the processes efficiently.

Marco Dorigo and Thomas Stutzle also pointed out in their book [6] that there is no general termination criterion for all existing metaheuristics. In practice only a number of rules of thumb are used for that purpose (see pp.47 in the book). Therefore, any global optimality condition is of both practical interests and theoretical importance. To the best of the author's knowledge, no general global optimality condition of polynomial-time complexity has ever been recognized so far for both combinatorial optimization and nonlinear programming. The study of the cooperative optimization metaheuristic attempts to provide an answer at certain degree to this important problem in the area of optimization.

2 Cooperative Optimization Metaheuristic

2.1 Cooperative Team Playing

Human beings, just like ants, are also part of nature. The former, just the later, also has highly structured social organizations. Human organizations can accomplish much more complex tasks than ant organizations. Therefore, human organizations can also provide computational models for solving difficult optimization problems. They should be as inspiring, at least not less, as ant organizations at discovering new optimization methods.

There is one important behavior in human organizations, the ability to achieve certain goals through cooperation. It happens at a variety of levels and very different scales. Cooperation is also a common social behavior of other species. However, it might not be easy to know exactly what other animals are thinking when they cooperate and how they cooperate. We might have better knowledge about ourselves when we cooperate with each other.

Usually, cooperation alone can hardly lead to a good solution either for an organization or for the individuals in the organization. Without competition, individuals in a organization may lose motivation to pursue better solutions even though they are willing to cooperate. On the other hand, without cooperation, individuals in an organization might directly conflict with each other and poor solutions might be reached both for the organization and themselves. Often times, through properly balanced competition and cooperation individuals in an organization can find the best solutions for the organization and possibly good solutions for themselves at the same time.

Team playing is a common cooperative behavior among human beings where the team members in a team work together through competition and cooperation to achieve the best for the team, but not necessarily the best for each member. Team playing can solve hard problems of complexity beyond the capability of any member in a team.

Preference passing and solution compromising are identified as two key strategies used in team playing. Preference passing is important in team playing in order that each team member knows the preferences of other team members on different solutions. Solution compromising enables the team members to resolve possible conflicts among them at picking the best solutions from their own perspectives.

Preference communicating and solution compromising are so common and pervasive as cooperation strategies that we might often take for granted. They play an important role for us to accomplish many challenging tasks. It is also possible to translate them into a working optimization method.

2.2 Cooperative Optimization and Mathematical Formalization

In the following discussions, for the generality of terms, we use "agent" to replace "team member" and "cooperative system" to replace "team" and "team playing". A agent can be a person, a neuron, a computer, a firm, an airplane, and many more. A cooperative system can consist of agents of the same kind or mixed.

To solve a hard optimization problem, we follow the divide-and-conquer principle. We first break up the problem into a number of sub-problems of manageable sizes and complexities. Following that, we assign each sub-problem to an agent in a cooperative system and ask those agents to solve the sub-problems in a cooperative way.

Each agent in the cooperative system finds the solution to the sub-problem assigned to it by applying a problem-specific optimization method or heuristic. Most of the time, the solution of each agent is not aligned with the rest of agents in the system. Solution compromising are required among the agents to resolve possible conflicts among them.

Some mathematical formalization is required from now on to make statements accurate. Given an objective function of n variables, $E(x_1, \ldots, x_n)$, or simply denoted as $E(x)$, our goal is to find a solution x^* to minimize $E(x)$,

$$x^* = \arg\min_x E(x) \ .$$

Assume that the domain for x_i is D_i, for $i = 1, 2, \ldots, n$. Assume further that we can break $E(x)$ into a linear aggregation of n sub-objective functions,

$$E(x) = E_1(x) + E_2(x) + \cdots + E_n(x) \ .$$

Let us assign each sub-objective function to an agent as its objective function in a cooperative system. There are thus n agents in the system thereafter. We let each agent to control one variable value, say the agent i responsible for picking a value for variable x_i. The goal of the system is to find a value for each variable such that $E(x)$ is minimized.

Usually, there are more than one variable contained in the objective function $E_i(x)$ of the agent i $(1 \leq i \leq n)$. To minimize its own objective function $E_i(x)$, the agent i can find the best value for each of those variables. However, it controls the value of only one variable, i.e., the value of the variable x_i. The values of the rest variables are controlled by other agents. The agents most often have conflicts at assigning values to variables. Two agents may not agree upon the best value for the same variable in terms of minimizing their own objective functions. We want the agents in the system to work together through competition, preference passing, and solution compromising such that the conflicts can be resolved and the global optimal solution x^* can be found.

Competition At the very beginning (t=0), each agent find the best value for its controlled variable by minimizing its own objective function,

$$\tilde{x}_i(t=0) = \arg\min_{x_i} \min_{X \setminus x_i} E_i(x) \ , \tag{1}$$

where X is the set of all variables, $X = \{x_1, x_2, \ldots, x_n\}$, and $X \setminus x_i$ denotes the set X minuses $\{x_i\}$.

Note that if the variable x_j is contained in the sub-objective function $E_i(x)$, it has an optimal value to minimize $E_i(x)$. This value may not be the same as the one to minimize $E_j(x)$, the objective function of the agent j. It may also not be the same as the one to minimize another objective function $E_{i'}(x)$ $(i' \neq i)$ containing x_j. This is called the conflicts in variable assignments.

Given a variable x_i, if its optimal values are the same for all sub-objective functions containing x_i, it is called a consensus is reached for assigning x_i. If a consensus assignment is reached for every variable, all those assignments form a solution, called a consensus solution, to the original minimization problem.

Because of the interdependence of the sub-objective functions $E_i(x)$, minimizing those sub-objective functions independently like (1) can hardly yield a consensus solution. For example, the assignment for x_i that minimizes E_i can hardly be the same as the assignment for the same variable that minimizes E_j if E_j contains x_i. Otherwise, the minimization problem is trivial to solve. Hence, agents in the system need to compromise their solutions so that a consensus in variable assignments can be reached.

Solution Preference Computing To cooperative with each other, every agents in the system needs to compute its own preferences on solutions and pass them to its neighbors. One way to achieve that is to let each agent i to compute the preferences of assigning different values for its controlled variable x_i at minimizing its own objective function $E_i(x)$. The preference measures can be represented by a real valued function defined on x_i, called the preference function. Given a variable value, $x_i \in D_i$, this function defines a numerical measure of the preference of assigning that value to the variable x_i at minimizing $E_i(x)$.

Note that the intermediate result in minimizing $E_i(x)$ (see (1)),

$$\Psi(x_i, t = 0) = \min_{X \setminus x_i} E_i(x) \ .$$

It defines a function on x_i, denoted as $\Psi_i(x_i, t = 0)$. Given a variable value $x_i = v$ ($v \in D_i$), the function value $\Psi_i(x_i = v, t = 0)$ shows the optimal result at minimizing $E_i(x)$ with x_i fixed at that particular value, $x_i = v$. The function has different function values for different x_i values, representing the differences of picking those variable values at minimizing $E_i(x)$.

Let us define a new function as

$$\psi_i(x_i, t = 0) = e^{-\Psi_i(x_i, t=0)} \ ,$$

If a variable value $x_i = v$ has a high function value $\psi_i(x_i = v, t = 0)$, it implies that assigning that value v to the variable x_i leads to a lower value of the objective function $E_i(x)$. The agent i prefers a variable value v_1 over another one v_2 if the function $\psi_i(x_i, t = 0)$ returns a higher function value for the former than that of the later, i.e.,

$$\psi_i(x_i = v_1, t = 0) > \psi_i(x_i = v_2, t = 0) \ .$$

Hence, this function can be served as a preference measure for the agent i at assigning a value to x_i, called the preference function. It also represents the soft decisions of the agent i at assigning the variable x_i. The function $\Psi_i(x_i, t = 0)$ (capital letter) is called the assignment constraint.

Preference Passing Here we do not assume that any pair of agents can communicate directly with each other. Every agent has certain neighbors and it can only pass its preference measures to its neighbors, denoted as $\mathcal{N}(i)$.

The agent i ($1 \leq i \leq n$) passes the assignment constraint Ψ_i instead of the preference function ψ_i to its neighbors. The technique reason will be given in the following solution compromising stage.

Solution Compromising One possible way to achieve solution compromising for the agent i, for $i = 1, 2, \ldots, n$, is to add the assignment constraints $\Psi_j(x_j, t = 0)$ received from its neighbors, to its own objective function $E_i(x)$ as its new objective function for the next iteration time $t = 1$, i.e.,

$$\tilde{E}_i(x, t = 1) = (1 - \lambda)E_i(x) + \lambda \sum_{j \in \mathcal{N}(i)} \Psi_j(x_j, t = 0) \ , \tag{2}$$

where λ is the coefficient for the linear combination, satisfying $0 < \lambda < 1$.

After all agents in the system modified their objective functions, they go back to the solution preference computing stage again to update their assignment constraints Ψ_i, for $i = 1, 2, \ldots, n$, based on the modified objective functions,

$$\begin{aligned}\Psi_i(x_i, t=1) &= \min_{X \setminus x_i} \tilde{E}_i(x, t=1) \\ &= \min_{X \setminus x_i} (1-\lambda) E_i(x) + \lambda \sum_{j \in \mathcal{N}(i)} \Psi_j(x_j, t=0) \end{aligned} \tag{3}$$

Such a process is repeated until all conflicts at variable assignments are resolved and a consensus solution is found, or the iteration time exceeds some cap.

The reason of passing the assignment constraints Ψ_i, for $i = 1, 2, \ldots, n$, instead of the preference functions ψ_i in the preference passing stage is to simplify mathematical manipulation. Otherwise, we need to take the exponent of the assignment constraint Ψ_i to get the preference function ψ_i in the preference passing stage, and take logarithm of the preference functions ψ_i to get back the assignment constraint Ψ_i in the solution compromising stage.

Minimizing the modified objective functions $\tilde{E}_i$ (see (3)) can possibly resolve the conflicts at variable assignments. Note that the modified objective function $\tilde{E}_i(x, t=1)$ defined in (2) will be dominated by $\Psi_j(x_j, t=0)$ if we choose a value for the parameter λ close to 1. In this case, for any x_j, $j \in \mathcal{N}(i)$, the optimal value for x_j to minimize $\tilde{E}_i(x, t=1)$, denoted as $\tilde{x}_j(\tilde{E}_i)$, tends to be the one of the minimal function value $\Psi_j(x_j, t=0)$, i.e.,

$$\tilde{x}_j(\tilde{E}_i) = \arg\min_{x_j} \Psi_j(x_j, t=0) \ .$$

It is equivalent to say that, given any x_j, $1 \leq j \leq n$, when we choose a value for λ close to 1, the optimal values of x_j to minimize all sub-objective functions that contain x_j tends to be of the same value $\arg\min_{x_j} \Psi_j(x_j, t=0)$. Therefore, conflicts in variable assignments tend to be resolved and a consensus solution is likely to be found in this case.

Obviously, the agent i may not be enable to achieve the best result at minimizing its original objective function $E_i(x)$ if it minimizes the modified objective function $\tilde{E}_i(x)$ instead. Therefore, it compromises its solution for resolving possible conflicts in variable assignments.

Parameter λ in (3) controls the level of cooperation and is called the cooperation strength. A small value for λ leads to weak cooperation among the agents in a cooperative system and a high value leads to strong cooperation among the agents. Obviously, strong cooperation is more referable than weak cooperation. As explained before, a cooperative system with strong cooperation tends to find a consensus solution.

Mathematical study shows that the update rule (2) for the modified objective functions $\tilde{E}_i(x)$ has a convergence problem. During each iteration, a copy of the assignment constraint $\Psi_i(x_i, t)$ of the agent i is sent to all its neighbors and added to their modified objective functions. The number of copies will be multiplied with an exponential rate with the increase of the iteration steps. To fix the problem, we break $\Psi_i(x_i, t)$ into several pieces and send out one piece to one agent. That is,

$$\Psi_i(x_i, t) \rightarrow w_{1,i}\Psi_i(x_i, t),\ w_{2,i}\Psi_i(x_i, t), \ldots,\ w_{n,i}\Psi_i(x_i, t) \ ,$$

where $w_{j,i}$, for $j = 1, 2, \ldots, n$, are non-negative weights satisfying $\sum_{j=1}^{n} w_{j,i} = 1$.

With that fix, the update rule for $\tilde{E}_i$ shown in Eq. (2) becomes

$$\tilde{E}_i(x, t=1) = (1-\lambda) E_i(x) + \lambda \sum_{j \in \mathcal{N}(i)} w_{ij} \Psi_j(x_j, t=0) \ , \tag{4}$$

2.3 Cooperative Optimization in a General Form

In the previous subsection, we described the detail of a possible way of translating a cooperative behavior into a working optimization method. Putting everything together, we have the cooperative optimization in the following general form.

Let $E(x_1, x_2, \ldots, x_n)$ be a multivariate objective function, or simply denoted as $E(x)$, where each variable x_i has a finite domain D_i. We break the function into n sub-objective functions $E_i(x)$ $(i = 1, 2, \ldots, n)$, one for each variable, such that E_i contains at least variable x_i, the minimization of each objective function $E_i(x)$ is computational manageable in complexity, and

$$E(x) = \sum_{i=1}^{n} E_i(x). \tag{5}$$

A cooperative optimization algorithm is defined by the following set of difference equations,

$$\Psi_i^{(k)}(x_i) = \min_{x_j \in X \setminus x_i} (1 - \lambda_k) E_i + \lambda_k \sum_j w_{ij} \Psi_j^{(k-1)}(x_j), \quad \text{for } i = 1, 2, \ldots, n, \tag{6}$$

where λ_k is the cooperation strength at the iteration step k and the superscript (k) is used for a more succinct form of stating the iteration step.

The pseudo code of a cooperative optimization algorithm is given as follows.

Procedure Cooperative Optimization Algorithm
1 Initialize the assignment constraints $(\Psi_i^{(0)}(x_i))$;
2 **for** $k := 1$ to *max_iteration* **do**
3 **for** each i **do**
4 **for** each $x_i \in D_i$ **do**
5 $\Psi_i^{(k)}(x_i) := \min_{X \setminus x_i}(1 - \lambda_k) E_i(x) + \lambda_k \sum_j w_{ij} \Psi_i^{(k-1)}(x_i)$;
6 $x := (\arg\min_{x_i} \Psi_i^{(k)}(x_i))$; /* update candidate solution */
7 **if** x is a consensus solution **return** x; /* as an optimal solution */
8 **return** x; /* as an approximate solution */

A cooperative optimization algorithm has excellent convergence properties. It has a unique equilibrium and converges to it with an exponential rate regardless of initial conditions and perturbations (see [2] for the detail and proofs). To be precise, the system of the difference equations (6) of cooperative optimization has one and only one fixed point. However, it is important to note that the equilibrium of a cooperative optimization algorithm is not necessarily a global optimum.

2.4 Further Improvement on the Basic Cooperative Optimization

The cooperative optimization method demonstrated superior performances at solving many practical optimization problems. In some cases, it is significantly better than many classic optimization methods. However, without further mathematical study, this cooperative optimization method will be just another black-box optimization method based on a set of empirical rules. Those rules may be very appealing and intuitively convincing at solving optimization problems. It is still a black-box optimization method with its performance varied from case to case and its overall performance is governed by the NFL theorems. We can not simply claim that this method is better than any other black box optimization method.

To open the black box of the cooperative optimization method, we need to know where to find a global optimum and whether a solution found by a cooperative optimization algorithm is a global optimum.

2.5 Cooperative Optimization and Global Optimality Conditions

The theoretical study has revealed an important property of the cooperative optimization method. In general, a consensus solution found by a cooperative optimization algorithm is the global optimal solution. Checking whether a consensus is reached for any variable is to check if the variable is of the same assigned value in all the modified sub-objective functions $\tilde{E}_i$ containing the variable. Such a test requires to check at most $n-1$ equalities. The checking for all variables takes at most $n(n-1)$ steps.

The cooperative optimization method offers a number of global optimality conditions of polynomial-time complexity for identifying global optimal solutions The paper [2] offers a number of them together with mathematical proofs.

The consensus checking of variable assignments also offers a direction for finding a global optimal solution. Since a consensus solution is a global optimal solution in general, the direction for finding a global optimal solution is to let agents in a cooperative system to possibly reach a consensus solution. A number of ways have been identified for that purpose.

The first way is to increase the cooperation strength λ_k. A higher value of λ_k leads to stronger cooperation among the agents in the system. As a consequence, each agent will weigh the solutions of other agents more than its own. Hence, a higher chance for those agents to resolve conflicts in variable assignments so that a consensus solution can be reached.

The second way is based on the variable clustering defined on a variable partition (detail in [1]). This technique can make the cooperative optimization method be complete, i.e., be always capable of finding the global optimum. The basic idea of the technique is to cluster any two variables, say x_i and x_j, if the agent i and the agent j have conflicts in assigning x_i, x_j, or both. The clustering is achieved by defining a new variable, denoted as (x_i, x_j), to replace x_i and x_j. The domain of the new variable is the Cartesian product of the domain of x_i and the domain of x_j, $D_i \times D_j$. Accordingly, we replace the assignment constraints associated with x_i and x_j, $\Psi_i(x_i)$ and $\Psi_j(x_j)$ by a new assignment constraint $\Psi_{i,j}(x_i, x_j)$ defined on the new variable. With that change, any conflicts in variable assignments between the agents i and j can be resolved. In analogy to team playing, if two persons in a team have conflicts in their decisions, we can let both of them to work closely with each other to straighten out their differences before they start to work with the rest of the team.

However, more variable clustering leads to a higher computational cost. At the extreme of variable clustering, a cooperative optimization algorithm becomes a brute-force search algorithm when all variables are clustered together. In this case, the algorithm guarantees to find a global optimum but will take a time exponential to the problem size.

With global optimality conditions, a cooperative optimization algorithm knows whether a solution it found at any given time is a global optimal solution. If it is not, it knows at which direction to find a global optimal solution. However, those statements do not imply that $NP{=}P$ because the cooperative optimization method can only verify within a polynomial time whether a solution it found is a global optimum or not. It cannot decide the global optimality for any given solution other than those it found. Second, although a cooperative optimization algorithm knows the direction to find global optimal solutions, it might take an exponential time to accomplish that over some set of optimization problems.

Despite of those limitations, a cooperative optimization algorithm can use the global optimality conditions to terminate its search process efficiently when it finds out that the solution it found at certain time is a global optimal solution. Furthermore, the directions suggested by those conditions can sometimes make its search process more effective than black box optimization algorithms. Often time, in practice, moving towards the directions take only a polynomial time before a global optimal solution is found.

Decoding low density parity check codes (LDPC) [20–24], for instance, is a NP-hard problem. The cooperative optimization has a successful application at solving this problem by applying the techniques mentioned above (see [4] for detail). In our HDTV lab at Tsinghua University, a

cooperative optimization algorithm has been tested using an experimental LDPC code. In this case, decoding that particular code can be formulated as an optimization problem with $7,493$ variables. When the signal-to-noise ratio $SNR \geq 2.7\ dB$, it has error rates less than 10^{-9} (one error out of one billion instances) at finding global optimal solutions. Its speed is over $2,000$ problem instances per second by a hardware implementation. This performance is good enough and fast enough for practical purposes. For this particular problem with $SNR = 2.7\ dB$, simulated annealing failed completely without finding any correct solution even though a significant amount of search time has been taken. Nevertheless, simulated annealing could be very useful in search for approximate solutions for other applications.

The LDPC decoding problem can be formulated as the following constrained optimization problem,

$$\min_{x_1,x_2,\ldots,x_n} \sum_{i=1}^{n} f_i(x_i), \quad \text{s.t. } \sum_{j=1}^{n} h_{ij}x_j = 0 \ mod\ 2 \ (1 \leq i \leq n) \ ,$$

where x_i are discrete variables of binary values, $x_i \in \{0,1\}$, for $1 \leq i \leq n$, $f_i(x_i)$ is a real-valued function defined on x_i, for $1 \leq i \leq n$, and h_{ij} are constants of binary values, $h_{ij} \in \{0,1\}$, for $1 \leq i,j \leq n$.

3 The Generality of Cooperative Optimization

Cooperative optimization requires to decompose an objective function into a linear aggregation of sub-objective functions. It turns out that a large class of optimization problems already have their objective functions in this form. Decomposing those objective functions is relatively straightforward in mathematics.

In general, those optimization problems have an objective function of the following form,

$$E(x_1, x_2, \ldots, x_n) = \sum_i f_{X_i}(X_i) \ , \tag{7}$$

where $f_{X_i}(X_i)$ is a real-valued component function defined on a subset of variables, $X_i \subset \{x_1, x_2, \ldots, x_n\}$.

If each component function in (7) involves at most two variables, the objective function becomes

$$E(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_i) + \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} f_{ij}(x_i, x_j) \ . \tag{8}$$

Many optimization problems that arose from many different fields have their objective functions in the form of (8). The examples are the famous traveling salesman problems (TSP), weighted maximum satisfiability problems, quadratic variable assignment problems, reasoning in Bayesian networks, network routing, stereo matching in computer vision, image segmentation in image processing, and many more. Furthermore, computational complexity theory tells us that any NP-complete problem can be transformed into another, say TSP, within a polynomial time. A good solution for TSP could help us to improve optimization techniques for solving other hard optimization problems.

One simple way to decompose the objective function (8) is given as follows

$$E_i = f_i(x_i) + \sum_{j,\ j\neq i} f_{ij}(x_i, x_j), \quad \text{for } i = 1, 2, \ldots, n \ . \tag{9}$$

If some component functions in (7) contain more than two variables in each function, we can cluster variables based on Cartesian product to transform them into binary functions or unary functions. Furthermore, the generalization of the decomposition given in (9) from binary functions to k-ary ($k \geq 2$) functions is straightforward in mathematics.

4 Limitations of Cooperative Optimization

The key requirement of cooperative optimization is to decompose an objective function into a linear aggregation of sub-objective functions. For those successful applications of cooperative optimization, a common characteristic of the decompositions is identified. Those decompositions have ideal problem instances that have the optimal solutions of sub-objective functions aligned with each other exactly. For any ideal problem instance, the solution of a sub-objective function is the restriction of the global optimal solution of the instance over the sub-set of variables that the function defines on. Other problem instances are distributed within a certain distance near those ideal problem instances in the problem instance space. For those problem instances, the cooperative optimization algorithms have remarkable performance.

Given an optimization problem with a known probabilistic distribution of problem instances, the above observation suggests to us that we need to discover an intrinsic decomposition dimension. Along the intrinsic decomposition dimension, the sub-objective functions have their optimal solutions aligned with each other within certain distance. If we decompose the objective function of the problem along the intrinsic dimension, we can solve the problem with a high successful rate.

There is no magic about a cooperative optimization algorithm at solving an optimization problem if its objective function decomposition is not along the intrinsic dimension or close to it. Its average performance might have no difference from other metaheuristic optimization methods at solving the problem. Sometimes, its performance could even be worse than some classic metaheuristic methods using a good problem-specific heuristic. The global optimality conditions possessed by the cooperative optimization method may have no use in the case simply because the solutions found by the cooperative optimization algorithm are not globally optimal for the most instances of the problem. Making the situation worse, the direction for finding global optimal solutions suggested by the global optimality conditions may be very inaccurate because sub-problem solutions are highly inconsistent with each other and provide no useful hint at the right search direction for the cooperative optimization algorithm.

5 Summary and Conclusions

The cooperation optimization method offers a a general framework of constructing algorithms to attack optimization problems. It first breaks a hard optimization problem into a number of sub-problems and solve them together in a cooperative way thereafter. At attacking an optimization problem, agents in a cooperative optimization system pass their preferences on solutions to their neighbors and try to resolve their conflicts in decisions through solution compromising. If all conflicts are eliminated, then a consensus solution is found, which is in general the global optimum of the optimization problem. If there are still unresolved conflicts after a certain number of iterations, agents can raise the level of cooperation among them to increase the chance to find a consensus solution. Another alternative is by putting agents with unresolved conflicts to work together closely. The cooperative optimization method can possibly identify global optimal solutions and know the directions of finding them. Those capabilities could sometimes offer this method some advantages in terms of efficiency and effectiveness. The computations of cooperative optimization systems are inherently distributed and parallel, making an entire system highly scalable and less vulnerable to perturbations and disruptions on agents than a centralized system.

References

1. Huang, X.: A general framework for constructing cooperative global optimization algorithms. In: Frontiers in Global Optimization. Nonconvex Optimization and Its Applications. Kluwer Academic Publishers (2004) 179–221

2. Huang, X.: Cooperative optimization for solving large scale combinatorial problems. In: Theory and Algorithms for Cooperative Systems. Series on Computers and Operations Research. World Scientific (2004) 117–156
3. Huang, X.: Image segmentation by cooperative optimization. In: IEEE International Conference on Image Processing (ICIP), Singapore (2004) 945–948
4. Huang, X.: Near perfect decoding of ldpc codes. In: Proceedings of IEEE International Symposium on Information Theory (ISIT). (2005) 302–306
5. Huang, X.: Cooperative optimization for energy minimization in computer vision: A case study of stereo matching. In: Pattern Recognition, 26th DAGM Symposium, Springer-Verlag, LNCS 3175 (2004) 302–309
6. Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge, Massachusetts, London, England (2004)
7. Eberhart, R.C., Shi, Y., Kennedy, J.: Swarm Intelligence. Morgan Kaufmann (2001)
8. Kirkpatrick, Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science **220** (1983) 671–680
9. Bilbro, G., *et al*, eds.: Optimization by mean field annealing. Morgan-Kauffman, San Mateo, CA (1989)
10. Hinton, G., Sejnowski, T., Ackley, D.: On theorectical foundations of a new algorithm for combinatorial optimization. Cognitive Science **9** (1985) 147–169
11. Geman, S., Geman, D.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-6** (1984) 721–741
12. Golver, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
13. Voudouris, C., Tsang, E.P.K.: Guided local search. Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK (1995)
14. Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Operaitons Research Leters **8** (1989) 67–71
15. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence through Simulated Evolution. John Wiley, New York (1966)
16. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA (1989)
17. Hinton, G., Sejnowski, T., Ackley, D.: Genetic algorithms. Cognitive Science (1992) 66–72
18. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
19. Pardalos, P., Resende, M.: Handbook of Applied Optimization. Oxford University Press, Inc. (2002)
20. Gallager, R.G.: Low-Density Parity-Check Codes. PhD thesis, Department of Electrical Engineering, M.I.T., Cambridge, Mass. (1963)
21. MacKay, D.J.C., Neal, R.M.: Good codes based on very sparse matrices. In: Cryptography and Coding, 5th IMA Conference. (1995)
22. Richardson, T.J., Shokrollahi, M.A., Urbanke, R.L.: Design of capacity-approaching irregular low-density parity-check codes. IEEE Transactions on Information Theory **47** (2001) 619–637
23. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
24. Kschischang, F.R., Frey, B.J., andrea Loeliger, H.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory **47** (2001) 498–519
25. Papadimitriou, C.H., Steiglitz, K., eds.: Combinatorial Optimization. Dover Publications, Inc. (1998)
26. Marr, D.: Vision:A computational investigation into the Human representation and processing of visual information. Freeman, San Francisco CA (1982)
27. Rosenfeld, A., Hummel, R., Zucker, S.: Scene labelling by relaxation operations. IEEE Transactions on System, Man, and Cybernetics **SMC-6** (1976) 420

28. Axelrod, R.: The Evolution of Cooperation. Basic Books (1984)
29. Zitnick, C.L., Kanade, T.: A cooperative algorithm for stereo matching and occlusion detection. IEEE TPAMI **2** (2000)
30. Hummel, R., Zucker, S.: On the foundations of relaxation labeling processes. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-5** (1983) 267
31. Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence **8** (1977) 99–118
32. Lawler, E.L., Wood, D.E.: Brand-and-bound methods: A survey. OR **14** (1966) 699–719
33. Jr., E.G.C., ed.: Computer and Job-Shop Scheduling. Wiley-Interscience, New York (1976)

Parallelization of a Fuzzy Heuristic using Multiple Search

Carlos Cruz, Juan R. González, and José L. Verdegay

Models of Decision and Optimization Research Group
Dept. of Computer Science & Artificial Intelligence
University of Granada
Granada, Spain
{carloscruz,jrgonzalez,verdegay}@decsai.ugr.es

Abstract. *Several types of parallel cooperation mechanisms may be built to find good solutions to difficult combinatorial optimization problems and to gain in terms of computation resources without a loss in solution quality. We propose a parallel strategy with a rules set based on fuzzy logic that has been applied successfully to combinatorial problems. We describe how these rules controls the search behavior of the threads. The knapsack problem and p-median problem serve as test cases. The procedures are coded in C++ using PVM in a shared memory machine.*

1 Introduction

Solving complex optimization problems requires high computational effort. One possibility proposed with increasing frequency in the last years to reduce the solving time is the use of parallel parallel versions of heuristics and metaheuristic methods. In the last decade, with the help of parallel and distributing processing, the research on multi-thread metaheuristics became a very active field. Several studies have shown that multi-thread techniques lead to better solutions than the corresponding sequential counterparts, even when the running time available to each thread is lower than that of the sequential computation. Studies have also shown that the combination of several threads, each one implementing a different strategy, increases the robustness of the global search relative to variations in the characteristics of problem instances [3, 4].

Following the classification of parallel metaheuristics of [3], our approach fitted the so called Type 3 parallelism or Multiple search: the strategy is made up of several solver agents that cooperate, by asynchronously exchanging information about the solutions they obtained, through a coordinator agent.

A relevant point in our implementation was the use of the most basic elements of Soft Computing [11]: fuzzy sets and fuzzy rules. A set of fuzzy rules are used at the coordinator level to control and modify the behavior of optimization algorithms, while fuzzy sets are used to define a fuzzy valuation in the threads to measure the degree to which the solutions that are considered at the decision stages, accomplishes a certain property.

These ideas also places our strategy within the context of fuzzy set-based heuristics for optimization [7, 10], whose main goal is to combine the best of fuzzy set and fuzzy system ideas with heuristic techniques in order to obtain robust, flexible and adaptive optimization tools.

The paper is organized as follows: Section 2 briefly describes the main characteristics of our strategy and introduces the fuzzy rules set. Section 3 describes the test problems and discusses the results of the computational experiments, and Section 4 concludes the work.

2 Cooperative Strategy based on fuzzy rules

The main idea of the strategy can be explained using the committee metaphor. Given some concrete problem to solve, we have a set or committee of solvers to deal with it. Each one applies a particular strategy to solve the problem in an independent fashion, and the whole set of solvers work simultaneously without direct interaction. In order to coordinate them, there is a central coordinator, who knows all the general aspects of the concerned problem and the particular features of solvers.

Figure 1 shows the scheme of the strategy. When the process starts, the Coordinator determines the initial behavior for every solver. When the solvers receive, they start to run. Periodically, the Coordinator receives reports from solvers with the partial results being obtained and sends them back some directives. The operation of the solvers is quite simple. They are running alone, receiving adaptation information from the coordinator and sending their performance. Both operations are controlled by conditions. The coordinator checks which solver provided new information and decides if its behavior needs to be adapted using a fuzzy rule base.

Each solver is implemented as a Fuzzy Adaptive Neighborhood Search method, FANS [6], which is essentially a local search framework, where the qualitative behavior of other simple local search techniques can be easily achieved. Besides, and being itself a local search technique, it is easy to understand, implement and does not need too much computational resources. By means of a fuzzy valuation $\mu()$, represented by a fuzzy set, a fuzzy measure of the generated solutions is obtained. Thus a degree of acceptability is calculated for each solution and such degrees are used by FANS at the decision stages.

In this work, the fuzzy valuation is based on the cost of the current or reference solution and represents a notion of acceptability. FANS moves between solutions satisfying the idea of acceptability with at least certain degree λ and variations in this parameter, enables the algorithm to achieve the qualitative behavior of other classical local search schemes [6]. This characteristic allows to build different search schemes and to drive diversification and intensification procedures easily. We therefore consider six possible values for λ, and we build a mixed scheme, as shown in Table 1, where the values of λ are different in order to incorporate both "greedy" and "relaxed" behaviors.

	Values of λ used in the Fuzzy Valuation					
Scheme	*Agent*1	*Agent*2	*Agent*3	*Agent*4	*Agent*5	*Agent*6
Mix	0.45	0.90	0.75	0.80	0.60	1.00

Table 1. Value of the λ parameter used in each agent to define a mixed scheme

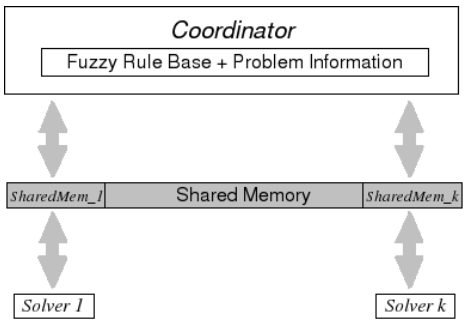


Fig. 1. Scheme of the optimization strategy

The Coordinator agent receives information from every solver agent. More precisely, each solver send: a time stamp, its current solution at that time and the cost of solution. After some transformations at the Coordinator, those values are stored in a memory of costs and a memory of improvements.

With these information, the Coordinator is able to decide if a solver agent is not doing well, analyzing the solver performance in terms of the history of values stored in both memories. The decision is made using the following rules:

1. **WM, Without Memory rule:**

IF the solution reported by $solver_i$ is **worse** than the best solution obtained **THEN** the *Coordinator* send the best solution to $solver_i$.

The *Solvers* communicate with the *Coordinator* whenever there is an improvement. If the *Solver* solution is better than the *Coordinator* one, then the *Coordinator* keeps this new best solution. In all other cases, the *Coordinator* sends its solution to the *Solver*.

2. **MR1, Memory Rule 1 (*Solver* independent):**

IF the quality of the solution reported by $solver_i$ is **low** *AND* the improvement rate of $solver_i$ is **low** **THEN** send a new solution to $solver_i$.

The label *low* is defined as follows:

$$\mu(x) = \begin{cases} 0 & \text{if } x > \beta \\ (\beta - x)/(\beta - \alpha) & \text{if } \alpha \leq x \leq \beta \\ 1 & \text{if } x < \alpha \end{cases} \quad (1)$$

where x is the percentile rank of a value (an improvement rate or a cost) in the samples stored in the coordinator memory, and the other parameters were fixed to $\alpha = 10$ and $\beta = 20$. The new solution could be a completely new one, for example generated randomly, or could be the best one saw by the coordinator until time t. This last option is the one implemented in this work which, in other words, may be understood as a greedy coordination scheme.
It should be noted that the first rule is independent of how the *Solvers* are defined, and can, in principle, be used when any local search type of *Solver* is being used.

3. **DR1, MR1 rule delayed with WM rule:**

IF computation time of $agente_i \leq \frac{1}{3}$ total computation time **THEN** uses WM rule **ELSE IF** computation time of $agente_i > \frac{1}{3}$ total computation time **THEN** uses MR1 rule.

where WM is used first throughout $\frac{1}{3}$ total computation time, and then scheme $MR1$ is applied.

4. **MR2, Memory Rule 2 (*Solver* dependent):**

IF the quality of the solution reported by $solver_i$ is **low**
AND the improvement rate of $solver_i$ is **low**
THEN make λ_i closer to λ^*.

where λ^* denotes the value of λ of the currently best performing *Solver*.
The consequent of Rule MR2, *make* λ_i *closer to* λ^*, is evaluated using a fuzzy label with the domain in $[\lambda_i, \lambda^*]$ if $\lambda_i \leq \lambda^*$, or vice versa. Under this approach, the λ_i value is changed as a linear function of the membership degree of the antecedent, either by increasing or decreasing its value.
In this case, the rule is FANS's specific. However, having been adjusted, the λ value can be considered as a change in the threshold accepting parameter. Taking this into account, this rule may be applied to any threshold-based *Solver*.

5. **DR2, delay MR2 rule:**

IF computation time of $agente_i \leq \frac{1}{3}$ total computation time
THEN apply WM rule
ELSE IF computation time of $agente_i > \frac{1}{3}$ total computation time
THEN apply MR2 rule.

where WM is used first throughout $\frac{1}{3}$ total computation time, and then scheme $MR1$ is applied.

3 Computational results

The goal of the experimental study is to evaluate the performance of the different strategies in terms of computational effort and solution quality.

Computational experiments have been performed on a cluster with 8 computers with Dual AMD Athlon processors at 2 GHz, 256 kbytes cache memory, 2 GBytes of RAM and O.S. Linux Fedora Core release 3. Tests have been run using 7 processors (6 solvers + 1 coordinator).

The asynchronous interactions between threads generally induce significant differences in search behaviour, not only for the global parallel method, but also for each search process participating to the cooperation. Therefore, the classical performance measures, speedup in particular, are not adequate to evaluate the performance of multi-search parallel meta-heuristics [1]. Thus, the performance in terms of solution quality is measured by error variable calculated with respect to the best solution known in the literature.

$$Error = 100 * \frac{Best\ Known - Obtained\ Value}{Best\ Known}$$

3.1 Knapsack Problem

First, we shall perform experiments on instances of the knapsack problem, and the mathematical formulation is:

$$Max \sum_{j=1}^{n} p_j \times x_j$$

$$s.t. \sum_{j=1}^{n} w_j \times x_j \leq C, x_j \in \{0,1\}, j = 1, \ldots, n$$

where n is the number of items, x_j indicates whether the item j is included or not in the knapsack, p_j is the profit associated with the item j, $w_j \in [0,..,r]$ is the weight of item j, and C is the capacity of the knapsack. We also assume that $w_j < C$, $\forall j$ (every item fits in the knapsack), and $\sum_{j=1}^{n} w_j > C$ (the whole set of items does not fit).

The choice of the knapsack as the test bed is based on the fact that we can construct test instances of varying hardness following three characteristics: size of the instance, type of correlation between weights and profits, and range of the values available for the weights. Although the knapsack is believed to be one of the "easiest" NP-hard problems, it is still possible to construct "hard to solve" instances that are considered as a challenge for most state-of-the-art exact algorithms [5, 8]. We shall deal with a set of instances generated and provided by Dr. David Pisinger, which belong to the following categories:

- Non-correlated Instances NC: $w_i = U(1,R), p_i = U(1,R)$.
- Weakly Correlated WC: $w_i = U(1,R), p_i = U(w_i - \frac{R}{10}, w_i + \frac{R}{10})$ with $p_i \geq 1$
- Strongly Correlated SC: $w_i = U(1,R), p_i = w_i + \frac{R}{10}$.
- Circle Instances, $circle(d = \frac{2}{3})$: $w_i = U(1,R), p = d\sqrt{(4R)^2 - (w - 2R)^2}$

The weights w_i are uniformly distributed in a given interval with the data range R. We randomly generate a series of H instances. The capacity of each instance h (of every type and value of R), with $h = 1 \ldots H$ is calculated as $c = \frac{h}{1+H} \sum_{i=1}^{n} w_i$. The optimum of each instance was also provided.

For the experiments, we have used instances with $n = (1000, 1500, 2000)$, and $R = 10000$. The *mixed* scheme has been used for the team of *Solvers*. The strategy is now run 10 times on each instance, and we gather the best values obtained at 5, 10, and 15 seconds.

The first analysis focused on Cases WM, MR1, and MR2. In this situation, the use of memory (Cases MR1 and MR2) did not contribute anything to the quality of the solutions (see Table 2). Moreover, after 5 seconds of computation, the errors achieved by the coordination with WM were almost unreachable for $MR1$ and $MR2$ even at the end of the run.

In our opinion, there is no balance between exploration and exploitation of the search space in these three coordination strategies; only exploitation is performed. In order to obtain a balanced strategy, we then defined Cases 4 and 5. In both cases, the use of memory is disabled during the first steps of the computation. More precisely, scheme WM is used during the first 5 seconds; after that, memory starts to be taken into account. Consequently, there is more exploration than exploitation at the beginning of the search. Meanwhile, the *Solvers* are reorganized in the search space in the (potentially) promising regions. When the memory is activated, tracking of the *Solvers* movements begins, which in turn enables the position of the *Solvers* to be further changed to regions where the best results are being obtained. Table 2 collects the average and typical deviation values for the error obtained after 15 seconds of computation on each type of instance.

The main thing we should highlight about the results is that the schemes with delayed use of memory almost always improve the average error with respect to the other schemes. In particular, scheme $DR2$ always obtains better results (on average) than the other schemes.

	NC		
Rule	1000	1500	2000
WM	3,73 (3,66)	2,87 (3,18)	3,61 (3,37)
MR1	3,92 (4,06)	3,10 (3,39)	3,68 (3,57)
MR2	3,00 (3,68)	2,72 (3,41)	3,57 (3,61)
DR1	3,85 (3,78)	3,09 (3,64)	3,61 (3,40)
DR2	3,33 (3,74)	2,65 (3,22)	3,25 (3,36)

	WC		
Rule	1000	1500	2000
WM	2,19 (1,59)	2,45 (1,59)	3,04 (1,95)
MR1	2,37 (1,72)	2,67 (1,77)	3,20 (2,09)
MR2	2,18 (1,61)	2,69 (2,10)	3,24 (2,32)
DR1	2,18 (1,52)	2,48 (1,59)	3,00 (1,91)
DR2	2,07 (1,56)	2,30 (1,57)	2,88 (1,86)

	SC		
Rule	1000	1500	2000
WM	1,96 (1,43)	1,90 (1,32)	2,32 (1,79)
MR1	2,10 (1,55)	2,15 (1,58)	2,65 (2,23)
MR2	2,43 (2,56)	2,41 (2,48)	2,96 (3,00)
DR1	1,92 (1,37)	1,92 (1,31)	2,35 (1,85)
DR2	1,86 (1,31)	1,80 (1,20)	2,23 (1,69)

	$Circle$		
Rule	1000	1500	2000
WM	11,86 (1,81)	10,22 (1,60)	9,79 (1,01)
MR1	12,06 (1,81)	10,41 (1,55)	10,00 (0,97)
M2R	11,46 (2,04)	9,99 (2,23)	10,16 (2,69)
DR1	11,87 (1,85)	10,17 (1,51)	9,84 (1,08)
DR2	11,45 (1,98)	9,60 (1,64)	9,20 (1,09)

Table 2. Average values of error and typical deviation for each rule and size

3.2 P-median Problem

Consider a set L of m potential locations for p facilities (or centers) and a set U of locations of n given users (or costumers, or demand points). The p-median problem (PM) is to locate simultaneously the p facilities at locations of L in order to minimize the total transportation cost for satisfying the demand of the users, each supplied from its closest facility. Consider also a $n \times m$ matrix D containing the distances travelled (or costs incurred) to satisfy the demand of the user located at i from the facility located at j, for all $j \in L$ and $i \in U$. In purely mathematical terms, the goal is to select p columns of D, such that the sum of minimum coefficients in each line within these columns is the smallest possible:

$$\min \sum_{i \in U} \min_{j \in J} d_{ij}$$

where, $J \subseteq L$ and $|J| = p$.

The distance matrix was taken from the TSPLIB's problem RL1400 [9] that includes 1400 customers (n). The values for the location sites were set to $p = \{10, 20, 30, \ldots, 100\}$ leading to 10 different problem instances. In our tests, the number of potential locations, m, is always equal to n.

The method was stopped on a maximum wall-clock time, $t_{max} = 240$ seconds. This value is similar to that used in [1] for the same distance matrix. Ten (10) repetitions have been performed for each rule and median.

Table 3 displays the best results obtained for different medians (p) and the Table 4 presents the number of cases (out of 10) with $error \leq 1$ for every test instance. We may consider these cases as successful runs because they ended with a low $error$. The *Total* row displays the sum of these values and, within parentheses, the number of different instances that were solved, at least once, with $error \leq 1$.

p	WM	MR1	DR1	MR2	DR2
10	0,00	0,00	0,00	0,00	0,00
20	0,23	0,03	0,08	0,00	0,08
30	0,54	0,33	0,45	0,53	0,67
40	0,71	0,55	0,30	0,56	0,30
50	0,84	0,70	0,69	0,89	0,81
60	1,90	0,68	1,16	1,03	1,27
70	1,49	1,18	1,46	0,68	1,11
80	1,87	1,08	1,97	1,43	1,21
90	1,55	1,04	0,80	0,93	0,92
100	1,84	0,97	1,78	1,07	1,09

Table 3. Minimum Values

p	WM	MR1	DR1	MR2	RR2
10	8	8	7	7	9
20	6	6	7	10	9
30	6	4	7	5	6
40	1	4	4	6	7
50	2	1	1	1	2
60	0	1	0	0	0
70	0	0	0	1	0
80	0	0	0	0	0
90	0	0	1	1	1
100	0	1	0	0	0
Total	23(5)	25(7)	27(6)	31(7)	34(7)

Table 4. Number of runs that ended with $error \leq 1$

First, we focus on the use of memory for coordination, so we will compare the results of column WM (rule without memory) vs. columns MR1 and MR2(rules with memory). The main fact we should highlight is that the schemes using memory always improve the best error with respect to the "without memory" rule. Moreover, they can obtain a higher number of cases with $error \leq 1$. We can conclude that the use of memory contributes making the strategy more robust, while improving the quality of the solutions obtained.

If we compare the results obtained by rule MR1 (with memory and agent's independent) vs. MR2 (with memory and agent's dependent), the differences are not relevant, although MR1 obtains slightly better results .

We proposed an additional scheme that delayed the use of memory for coordination to the last 2/3 of the time available. In the first 1/3 of the time, the agents may run completely independent or they may be coordinated with the WM rule.

So, the next analysis is aimed to check if this proposed delay is beneficial or not, and if the answer is yes, which strategy is best suited to be applied in the first stage of the run.

First, we look at DR1 with respect to MR1 results. In terms of the best values, it seems that as the value of p increases, MR1 obtains lower error values. When looking at the number of successful runs, MR1 and DR1 obtain similar rate of success.

Looking at DR2 with respect to MR2, we have no evidence to claim if the use of delay is better than a "no delay" scheme neither in terms of the best values, but DR2 obtains a higher number of success than MR2 and therefore more robust performance.

About the whole set of rules, the scheme DR2 with delayed use of memory obtains the highest number of successful runs. In this rule, memory and cooperation are disabled at the beginning of the search, letting the solver to be more explorers than exploiters. Later on, when the MR2 rule is activated and the solvers are starting to converge, memory starts to work and solvers are repositioned in the most promising regions of the search space. In short, the strategy with this rule is able to achieve a global balance between exploration and exploitation of the search space.

4 Conclusions

In this article, we have analyzed the behavior of a cooperative multi-thread based strategy, where each solver is modelled as an optimization algorithm, executed asynchronously, and controlled by a coordinator. One relevant point is the use of memory to keep track of both the state of the individual threads and the obtained solutions. Based on this memory, a set of fuzzy rules is used to control the system behavior. The use of memory in the context of parallel metaheuristics was recently proposed as a "challenge" in [2], and our work, proposing the jointly use of memory and fuzzy rules, should therefore be considered, as far as we are aware, as a novel way to face such a challenge.

The coordination schemes defined here give rise to optimization behaviors that are more directed towards the refinement of good solutions, which are therefore suitable for application during

the exploitation phase, rather than towards exploration (which may occur at the beginning of the search). The results indicate that the use of cooperative strategies, coupled with memory and fuzzy rules, obtains better results than the no memory rule, and that the schemes with delayed use of memory almost always improve the results with respect to the other schemes.

Acknowledgments

This work was partially funded by TIN-2005-08404-C04-01 and TIC-00129-JA.

References

1. T.G. Crainic, M. Gendreau, P. Hansen, and N. Mladenovic. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics*, 10:293–314, 2004.
2. T.G. Crainic and M. Toulouse. *Fleet Management and Logistics*, chapter Parallel Metaheuristics. Kluwer Academic, 1998.
3. T.G. Crainic and M. Toulouse. *Parallel Strategies for Metaheuristics*, volume Handbook of Metaheuristics, pages 475 – 513. Kluwer Academic Publisher, 2003.
4. V. Cung, S.L. Martins, C. Ribeiro, and C. Roucairol. *Essays and Surveys in Metaheuristics*, chapter Strategies for the Parallel Implementation of Metaheuristics, pages 263–308. Kluwer Academic Publisher, 2001.
5. H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2004.
6. D. Pelta, A. Blanco, and J.L. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.
7. D. Pelta, C. Cruz, A. Sancho-Royo, and J.L. Verdegay. Using memory and fuzzy rules in a co-operative multi-thread strategy for optimization. *Information Sciences*, 176:1849–1868, 2006.
8. D. Pisinger. Where are the hard knapsack problems? Technical report, University of Copenhagen, DIKU, Denmark, 2003/08.
9. G. Reinelt. Tsplib: A traveling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.
10. J.L. Verdegay, editor. *Fuzzy Sets based Heuristics for Optimization*. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003.
11. L.A. Zadeh. Fuzzy logic, neural networks, and soft computing. *Commun. ACM*, 37(3):77–84, 1994.

Cell-like and Tissue-like Membrane Systems as Recognizer Devices

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, and Francisco J. Romero-Campero

Research Group on Natural Computing
Dpt. Computer Science and Artificial Intelligence, University of Sevilla, Sevilla, Spain
ETS Ingeniería Informática, Avda. Reina Mercedes s/n
{magutier,marper,ariscosn,fran}@us.es

Abstract. *Most of the variants of membrane systems found in the literature are generally thought as generating devices. In this paper recognizer computational devices (cell–like and tissue–like) are presented in the framework of Membrane Computing, using the biological membranes arranged hierarchically, inspired from the structure of the cell, and using the biological membranes placed in the nodes of a graph, inspired from the cell inter–communication in tissues. In this context, polynomial complexity classes of recognizer membrane systems are introduced. The paper also addresses the* **P** *versus* **NP** *problem, and the (efficient) solvability of computationally hard problems, in the framework of these new complexity classes.*

1 Introduction

One of the main goals of a computing model is to solve problems. In order to design computational devices capable of attacking decision problems, we must decide how to represent by strings the instances of the problem. In that context, to solve a decision problem consists of recognizing the language associated with it.

Membrane Computing is a young branch of Natural Computing providing distributed parallel computing models whose computational devices are called *membrane systems*, which are inspired by some basic biological features, by the structure and functioning of the living cells, as well as from the cooperation of cells in tissues, organs, and organisms.

In this area there are basically two ways to consider computational devices: cell–like membrane systems and tissue–like membrane systems. The first one, using the biological membranes arranged hierarchically, inspired from the structure of the cell, and the second one using the biological membranes placed in the nodes of a graph, inspired from the cell inter–communication in tissues.

In this paper we present recognizer membrane systems (both cell–like and tissue–like variants) as a framework to address ways to efficiently solving computationally hard problems, capturing the true concept of algorithm in spite of providing a non–deterministic computing model.

2 Preliminaries

The computational devices of a model of computation are designed to handle inputs and outputs that are strings over a finite alphabet.

Usually, **NP**-completeness has been studied in the framework of *decision problems*, but it is not an important restriction because one can easily transform any optimization problem into a

roughly equivalent decision problem by supplying a target value for the quantity to be optimized, and asking the question whether this value can be attained.

Definition 1. A decision problem is a pair (I, θ) such that I is a language over a finite alphabet (whose elements are called *instances*) and θ is a total boolean function (that is, a predicate) over I.

There exists a natural correspondence between languages and decision problems in the following way. Each language L, over an alphabet Σ, has a decision problem, X_L, associated with it as follows: $I_{X_L} = \Sigma^*$, and $\theta_{X_L} = \{(u, 1) \mid u \in L\} \cup \{(u, 0) \mid u \in \Sigma^* - L\}$; reciprocally, given a decision problem $X = (I_X, \theta_X)$, the language L_X over the alphabet of I_X corresponding to it is defined as follows: $L_X = \{u \in I_X \mid \theta_X(u) = 1\}$.

The **P** versus **NP** problem is the problem of determining whether every problem solvable by some non-deterministic Turing machine in polynomial time can also be solved by some deterministic Turing machine in polynomial time. It is one of the outstanding open problems in theoretical computer science. A negative answer to this question would confirm that the majority of current cryptographic systems are secure from a practical point of view. A positive answer would not only show the uncertainty about the security of these systems, but also this kind of answer is expected to come together with a general procedure that provides a deterministic algorithm solving most of the **NP**–complete problems in polynomial time.

In the last years several computing models using powerful tools from nature have been developed (because of this, they are known as *bio-inspired* models) and several solutions in polynomial time to problems from the class **NP** have been presented, making use of non-determinism and/or of an exponential amount of space. This is the reason why a practical implementation of such models (in biological, electronic, or other media) could provide a significant advance in the resolution of computationally hard problems.

3 Cell–like recognizer membrane systems

In the structure and functioning of a cell, biological *membranes* play an essential role. The cell is separated from its environment by means of a *skin membrane*, and it is internally compartmentalized by means of *internal membranes*.

The main *syntactic* ingredients of a cell–like membrane system are the *membrane structure*, the *multisets*, and the *evolution rules*.

- A *membrane structure* consists of several membranes arranged in a hierarchical structure inside a main membrane (the *skin*), and delimiting *regions* (the space in–between a membrane and the immediately inner membranes, if any). Each membrane identifies a region inside the system. A membrane structure can be considered as a rooted tree.
- Regions defined by a membrane structure contain objects corresponding to chemical substances present in the compartments of a cell. The objects can be described by symbols or by strings of symbols, in such a way that *multiset of objects* are placed in regions of the membrane structure.
- The objects can evolve according to given *evolution rules*, associated with the regions (hence, with the membranes).

The *semantics* of the cell–like membrane systems is defined through a non deterministic and synchronous model (in the sense that a global clock is assumed) as follows:

- A *configuration* of a cell–like membrane system consists of a membrane structure and a family of multisets of objects associated with each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system.

- In each time unit we can transform a given configuration in another configuration by applying the evolution rules to the objects placed inside the regions of the configurations, in a non–deterministic, and maximally parallel manner (the rules are chosen in a non–deterministic way, and in each region all objects that can evolve must do it). In this way, we get *transitions* from one configuration of the system to the next one.
- A *computation* of the system is a (finite or infinite) sequence of configurations such that each one is obtained from the previous one by a transition, and shows how the system is evolving.
- A computation which reaches a configuration where no more rules can be applied to the existing objects, is called a *halting computation*.
- The result of a halting computation is usually defined through the multiset associated with a specific output membrane (or the environment) in the final configuration.

In the basic version, cell–like membrane systems can be seen as generating devices, working in a non–deterministic and maximally parallel manner, with output membrane, and without input membrane.

But we are very interested in to use cell-like membrane systems in order to solve decision problems. What are the necessary ingredients to solve problems with a computational device?

We will see that we must work with non–deterministic (but confluent) systems, using maximal parallelism, without output membrane (the output will be in the enviroment), and with input membrane.

Definition 2. A *P system with input* is a tuple (Π, Σ, i_Π), where: (a) Π is a P system, with working alphabet Γ, with p membranes labelled by $1, \ldots, p$, and initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_p$ associated with them; (b) Σ is an (input) alphabet strictly contained in Γ and the initial multisets are over $\Gamma - \Sigma$; and (c) i_Π is the label of a distinguished (input) membrane.

If m is a multiset over Σ, then the *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots \mathcal{M}_p)$.

Definition 3. A *cell–like recognizer membrane system* is a P system with input, (Π, Σ, i_Π), and with external output such that: (1) The working alphabet contains two distinguished elements YES, NO; (2) All computations halt; and (3) In every computation of Π, either some object YES or some object NO (but not both) must have been released into the environment, and only in the last step of the computation.

We say that $\mathcal{C}$ is an accepting (respectively, rejecting) computation if the object YES (respectively, NO) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.

We denote by $\mathcal{R}$ the class of all cell–like recognizer membrane systems.

We propose to solve a decision problem through a family of P systems (constructed in a uniform way) such that each element of the family processes all the instances of *equivalent size*, in some sense (we say that these solutions are *uniform* solutions).

Next, we define what means to solve a decision problem in the framework of cell–like membrane systems, and in an uniform way.

Definition 4. We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$, of $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}_\mathcal{R}$, if the following is true:

- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines; that is, there exists a deterministic Turing machine constructing $\Pi(n)$ from $n \in \mathbf{N}$ in polynomial time.
- There exists a pair (cod, s) of polynomial-time computable functions whose domain is L, such that:
  - For each $u \in L$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of $\Pi(s(u))$.

- The family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(h(u))$ with input $g(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps.
- The family $\mathbf{\Pi}$ is sound, with regard to (X, cod, s); that is, for each $u \in I_X$ it is verified that if there exists an accepting computation of $\Pi(h(u))$ with input $g(u)$, then $\theta_X(u) = 1$.
- The family $\mathbf{\Pi}$ is complete with regard to (X, cod, s); that is, for each $u \in I_X$ it is verified that if $\theta_X(u) = 1$, then every computation of $\Pi(h(u))$ with input $g(u)$ is an accepting one.

In the above definition we have imposed every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation with the *same* input produces the *same* output.

We have the class $\mathbf{PMC}_{\mathcal{R}}$ is closed under polynomial–time reduction and complement.

If systems without input membrane are used, constructing *one* specific system for each instance, but keeping the 'polynomially uniform by Turing machines' condition, then we say that the obtained solutions are *semi–uniform*. We shall denote by $\mathbf{PMC}^*_{\mathcal{R}}$ the class of problems solvable in polynomial time by a semi–uniform family of systems in $\mathcal{R}$.

4 The P versus NP problem in the context of cell–like recognizer membrane systems

We consider deterministic Turing machines as language recognizer devices. Then, we can associate with each deterministic Turing machine a decision problem, which will permit us to define when such a machine is simulated by a family of P systems (this issue was also addressed e.g. in [12, 20]).

Definition 5. Let M be a Turing machine with input alphabet Σ_M. The *decision problem associated with* M is the problem $X_M = (I, \theta)$, where $I = \Sigma^*_M$, and for every $w \in \Sigma^*_M$, $\theta(w) = 1$ if and only if M accepts w.

Obviously, the decision problem X_M is solvable by the Turing machine M.

Definition 6. We say that a Turing machine, M, is simulated in polynomial time by a family of systems of the class $\mathcal{R}$, if $X_M \in \mathbf{PMC}_{\mathcal{R}}$.

In cell–like membrane systems, evolution rules, communication rules and rules involving dissolution are called *basic rules*. That is, by applying this kind of rules the size of the membrane structure does not increase. Hence, it is not possible to construct an exponential working space in polynomial time using only basic rules in a cell–like membrane system.

We recall here a result from Chapter 9 of [22].

Proposition 1. *Let M be a deterministic Turing machine working in polynomial time. Then M can be simulated in polynomial time by a family of cell–like recognizer membrane systems using only basic rules.*

Reciprocally, in [20] the following result was proved:

Proposition 2. *For every decision problem solvable in polynomial time by a family of cell–like recognizer membrane systems using only basic rules, there exists a Turing machine solving it in polynomial time.*

Under the hypothesis $\mathbf{P}{\neq}\mathbf{NP}$, Zandron et al. [23] established the limitations of cell-like membrane systems which use only basic rules concerning the efficient solution of **NP**-complete problems. This result was generalized by Pérez–Jiménez et al. [20] obtaining the following two characterizations of the $\mathbf{P} \neq \mathbf{NP}$ relation by means of unsolvability results in polynomial time for **NP**–complete problems by families of cell–like recognizer membrane systems using only basic rules.

Theorem 1. *The following propositions are equivalent:*

1. $\mathbf{P} \neq \mathbf{NP}$.
2. *There exists an* $\mathbf{NP}$*–complete decision problem unsolvable in polynomial time by a family cell–like recognizer membrane systems using only basic rules.*
3. *Each* $\mathbf{NP}$*–complete decision problem is unsolvable in polynomial time by a family of cell–like recognizer membrane systems using only basic rules.*

Let us denote by $\mathcal{RB}$ the class of cell–like recognizer membrane systems using only basic rules. From the constructive proof given in [22], we deduce the following result characterizing the standard complexity class $\mathbf{P}$.

Theorem 2. $\mathbf{P} = \mathbf{PMC}_{\mathcal{RB}}$.

5 Recognizer cell–like membrane systems with active membranes

A particularly interesting class of cell–like membrane systems are the systems with active membranes, where the membrane division can be used in order to solve computationally hard problems, e.g., **NP**-complete problems, in polynomial or even linear time, by a space–time trade-off.

Definition 7. A recognizer cell–like membrane system with active membranes is a recognizer cell–like membrane system (Π, Σ, i_Π) where the rules of the associated P system are of the following forms (H being the set of labels of Π):

1. $[\, a \to \omega\,]_h^\alpha$ for $h \in H$, $\alpha \in \{+, -, 0\}$, $a \in \Sigma$, $\omega \in \Sigma^*$: An object a within a membrane labelled with h and polarity α, evolves to a multiset ω.
2. $a\,[\ \,]_h^{\alpha_1} \to [\, b\,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$: An object from the region immediately outside a membrane labelled with h is introduced in this membrane, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.
3. $[\, a\,]_h^{\alpha_1} \to b\,[\ \,]_h^{\alpha_2}$ for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Sigma$: An object is sent out from membrane labelled with h to the region immediately outside, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.
4. $[\, a\,]_h^\alpha \to b$ for $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in \Sigma$: A membrane labelled with h is dissolved in reaction with an object. The skin is never dissolved.
5. $[\, a\,]_h^{\alpha_1} \to [\, b\,]_h^{\alpha_2}\ [\, c\,]_h^{\alpha_3}$ for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in \Sigma$: An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.
6. $[\ [\ ]_{h_1}^{\alpha_1} \ldots [\ ]_{h_k}^{\alpha_1}\ \ [\ ]_{h_{k+1}}^{\alpha_2} \ldots [\ ]_{h_m}^{\alpha_2}\ ]_{h_0}^{\alpha_0} \quad \to \quad [\ [\ ]_{h_1}^{\alpha_3} \ldots [\ ]_{h_k}^{\alpha_3}\ ]_{h_0}^{\alpha_5}\ \ [\ [\ ]_{h_{k+1}}^{\alpha_4} \ldots [\ ]_{h_m}^{\alpha_4}\ ]_{h_0}^{\alpha_6}$, for $k \geq 1$, $m > k$, $h_i \in H$ for $0 \leq i \leq m$, and $\alpha_1, \ldots, \alpha_6 \in \{+, -, 0\}$, with $\{\alpha_1, \alpha_2\} = \{+, -\}$. These are division rules for non–elementary membranes. If the membrane with label h_0 contains other membranes than those with labels $h_1, \ldots, h_m$, then they must have neutral charge in order to make this rule applicable; these membranes and their contents are duplicated and placed in both new copies of the membrane h_0. Besides, every object in region h_0, as well as all membranes and objects placed inside membranes $h_1, \ldots, h_m$, are reproduced in the new copies of membrane h_0.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object which can evolve by one rule of any form, must evolve.

- If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.
- If at the same time a membrane labelled by h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.
- The rules associated with membranes labelled by h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)-(e).

Let us denote by $\mathcal{AM}$ the class of recognizer P systems with active membranes using 2-division. Different polynomial time solutions for **NP**–complete problems have been obtained using this class of cell–like recognizer membrane systems: *Knapsack* ([14]), *Subset Sum* ([13]), *Partition* ([4]), *SAT* ([19]), *Clique* ([1]), *Bin Packing* ([15]), and *CAP* ([16]).

Having in mind that the complexity class $\mathbf{PMC}_{\mathcal{AM}}$ is closed under complement and polynomial time reductions we have the following result.

Proposition 3. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$, *and* $\mathbf{co\text{-}NP} \subseteq \mathbf{PMC}_{\mathcal{AM}}$.

The complexity class $\mathbf{PMC}_{\mathcal{AM}}$ does not seem precise enough to describe classical complexity classes below **NP**. Therefore, it is challenging to investigate weaker variants of cell–like membrane systems able to characterize classical complexity classes.

In [2] universality has been achieved by removing the polarization of membranes from P systems with active membranes but allowing the change of membrane labels.

Several efficient solutions to **NP**–complete problems have been obtained within the following variant of membrane systems with active membranes:

- P systems using 2–division for elementary membranes, without cooperation, without priorities, without label changing, but using only two electrical charges (A. Alhazov [2], A. Riscos [21]).
- P systems using 2–division for elementary membranes, without cooperation, without priorities, without label changing, without polarizations, but using bi–stable catalysts (M.J. Pérez and F.J Romero [17]).
- P systems without polarizations, without cooperation, without priorities, without label changing, without division, but using three types of membrane rules: separation, merging and release (L. Pan et al. [7]).
- P systems with separation rules instead of division rules, in two different cases: (a) using polarizations and separation rules; and (b) without polarizations, but using separation rules with change of membrane labels (L. Pan and T.O. Ishdorj [8]).

It is possible to obtain polynomial time solutions to **NP**–complete problems through cell–like recognizer membrane systems with active membranes using 2-division for elementary membranes. But, what happens if we remove polarizations? We denote by $\mathcal{AM}^0$ the class of this kind of recognizer P systems.

Question: What is exactly the class of decision problems solvable in polynomial time by families of systems belonging to $\mathcal{AM}^0$?

We denote by $\mathcal{AM}^0(\alpha,\beta)$, where $\alpha \in \{-d,+d\}$ and $\beta \in \{-ne,+ne\}$, the class of all cell–like recognizer P systems with polarizationless active membranes such that: (a) if $\alpha = +d$ (resp. $\alpha = -d$) then dissolution rules are permitted (resp. forbidden); and (b) if $\beta = +ne$ (resp. $\beta = -ne$) then division rules for elementary and non–elementary (resp. only division rules for elementary) membranes are permitted.

Proposition 4. *For each* $\alpha \in \{-d,+d\}$ *and* $\beta \in \{-ne,+ne\}$ *we have:*

(1) $\mathbf{PMC}_{\mathcal{AM}^0(\alpha,\beta)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,\beta)}$.

(2) $\mathbf{PMC}_{\mathcal{AM}^0(\alpha,-ne)} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(\alpha,+ne)}$.
(3) $\mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,-ne)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(\alpha,+ne)}$.
(4) $\mathbf{PMC}_{\mathcal{AM}^0(-d,\beta)} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(+d,\beta)}$.
(5) $\mathbf{PMC}^*_{\mathcal{AM}^0(-d,\beta)} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(+d,\beta)}$.

In the framework of recognizer P systems with membrane division but without using polarizations it has been shown a surprising role of the dissolution rules, as it makes the difference between efficiency and non–efficiency for P systems with membrane division and without polarization ([3, 5]).

Theorem 3. *We have the following:*

(1) $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0(-d,\beta)} = \mathbf{PMC}^*_{\mathcal{AM}^0(-d,\beta)}$, *for each* $\beta \in \{-ne, +ne\}$.
(2) $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0(+d,+ne)}$.
(3) $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{AM}^0(+d,+ne)}$.

6 Tissue–like recognizer membrane systems with active membranes

In this section we consider computational devices inspired from the cell inter–communication in tissues, and adding the ingredient of cell division rules of the same form as in cell–like membrane systems with active membranes, but without using polarizations.

In these systems, the rules are used in the non-deterministic maximally parallel way, but we suppose that when a cell is divided, its interaction with other cells or with the environment is blocked; that is, if a division rule is used for dividing a cell, then this cell does not participate in any other rule, for division or communication. The set of communication rules implicitely provides the graph associated with the system through the labels of the membranes. The cells obtained by division have the same labels as the mother cell, hence the rules to be used for evolving them or their objects are inherited.

Definition 8. A *tissue–like membrane system with active membranes* is a tuple

$$\Pi = (\Gamma, \Sigma, \mathcal{M}_1, \ldots, \mathcal{M}_p, E, R, i_{in}),$$

where:

1. $p \geq 1$ (the initial degree of the system; the system contains p cells, labelled with $1, 2, \ldots, p$);
2. Γ is the working alphabet containing two distinguished objects *YES* and *NO*;
3. Σ is an (input) alphabet strictly contained in Γ.
4. $\mathcal{M}_1, \ldots, \mathcal{M}_p$ are multisets over $\Gamma - \Sigma$, describing the objects placed in the cells of the system (we suppose that at least one copy of *YES* and *NO* is in some of these multisets);
5. $E \subseteq \Gamma$ is the set of objects present in the environment in arbitrary many copies each (the objects *YES* and *NO* are not present in E);
6. R is a finite set of *developmental rules*, of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, p\}, i \neq j$, and $u, v \in \Gamma^*$; $1, 2, \ldots, p$ identify the cells of the system, 0 is the environment: When applying a rule $(i, x/y, j)$, the objects of the multiset represented by u are sent from region i to region j and simultaneously the objects of the multiset v are sent from region j to region i;
   (b) $[\,a\,]_i \rightarrow [\,b\,]_i[\,c\,]_i$, where $i \in \{1, 2, \ldots, p\}$ and $a, b, c \in \Gamma$: Under the influence of object a, the cell with label i is divided in two cells with the same label; in the first copy the object a is replaced by b, in the second copy the object a is replaced by c; all other objects are replicated and copies of them are placed in the two new cells.

7. $i_{in} \in \{1, \ldots, n\}$ is the label of the input membrane.

Let m be a multiset over Σ. The *initial configuration of Π with input m* is the tuple $(\mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} \cup m, \ldots, \mathcal{M}_p)$.

The rules of a tissue–like membrane system as above are used in the non-deterministic maximally parallel manner as customary in membrane computing. In each step, we apply a set of rules which is maximal (no further rule can be added), with the following important restriction: if a cell is divided, then the division rule is the only one which is applied for that cell in that step, its objects do not participate in any communication rule.

The computation starts from the initial configuration and proceeds as defined above; only halting computations give a result, and the result is given by the presence of a distinguished object in the environment.

Definition 9. A tissue–like membrane system with active membranes Π is a recognizer system if: (a) all computations halt; and (b) in every computation of Π, either a copy of the object *YES* or a copy of the object *NO* (but not both) is sent into the environment.

We say that $\mathcal{C}$ is an accepting (rejecting) computation if the object *YES* (respectively, the object *NO*) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.

We denote by $\mathcal{TR}$ the class of tissue–like recognizer membrane systems with active membranes.

In order to present the concept of uniform solvability in the framework of membrane systems as above, we define the concept of polynomial encoding from a language in a family of tissue–like recognizer membrane systems with active membranes.

Definition 10. Let L be a language, and $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ a family of tissue–like recognizer membrane systems of $\mathcal{TR}$. A polynomial encoding of L in $\mathbf{\Pi}$ is a pair (cod, s) of polynomial-time computable functions whose domain is L, and for each $u \in L$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$.

In the present paper we are interested in the computing efficiency. That is why we have introduced a variant of tissue–like systems with membrane division.

Next we define the concept of solvability in this new framework and in a similar way to Definition 4.

Definition 11. We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$, of tissue–like recognizer membrane systems with active membranes, and we denote this by $X \in \mathbf{PMC}_{\mathcal{TR}}$, if the following is true:

- The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.
- There exists a polynomial encoding (cod, s) from I_X to $\mathbf{\Pi}$ such that the family $\mathbf{\Pi}$ is polynomially bounded, sound and complete with regard to (X, cod, s).

We also have the class $\mathbf{PMC}_{\mathcal{TR}}$ is closed under polynomial–time reduction and complement.

We have said nothing about the way the computations proceed; in particular, they can be non-deterministic, as standard in membrane computing. It is important however to remark that the systems always stop and they always send out an object which is the correct answer to the instance of the problem that they are processing.

This natural extension of tissue P systems provides the possibility of solving `SAT` in polynomial time, in a confluent way: at precise times, one of the objects YES, NO is sent to the environment, giving the answer to the question whether the input propositional formula is satisfiable. We refer to [11] for a more detailed presentation of the following design.

Let us consider a propositional formula $\varphi = C_1 \wedge \cdots \wedge C_m$, consisting of m clauses $C_j = y_{j,1} \vee \cdots \vee y_{j,k_j}$, where $y_{j,i} \in \{x_l, \neg x_l \mid 1 \leq l \leq n\}$ (there are used n variables). Without loss of generality, we may assume that no clause contains two occurrences of some x_i or two occurrences

of some $\neg x_i$ (the formula is not redundant at the level of clauses), or both x_i and $\neg x_i$ (otherwise such a clause is trivially satisfiable, hence can be removed).

We consider the family $\mathbf{\Pi} = \{\Pi(\langle n, m\rangle) : n, m \in \mathbf{N}\}$ of tissue–like recognizer membrane systems, being $\langle n, m\rangle = \frac{(n+m)\cdot(n+m+1)}{2} + n$.

The tissue–like recognizer membrane system

$$\Pi(\langle n, m\rangle) = (\Gamma(\langle n, m\rangle), \Sigma(\langle n, m\rangle), \mathcal{M}_1, \mathcal{M}_2, E(\langle n, m\rangle), R(\langle n, m\rangle), i_{in})$$

will process all Boolean formulae in conjunctive normal form with n variables and m clauses, and is defined as follows:

$$\begin{aligned}
\Gamma(\langle n, m\rangle) &= \{a_i, t_i, f_i \mid 1 \leq i \leq n\} \cup \{r_i \mid 1 \leq i \leq m\} \\
&\cup \{T_{i,1}, F_{i,1} \mid 1 \leq i \leq n\} \cup \{T'_{i,j}, F'_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m+1\} \\
&\cup \{s_{i,j}, s'_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\} \\
&\cup \{b_i \mid 1 \leq i \leq 3n+m+1\} \cup \{c_i \mid 1 \leq i \leq n+1\} \\
&\cup \{d_i \mid 1 \leq i \leq 3n+nm+m+2\} \cup \{e_i \mid 1 \leq i \leq 3n+nm+m+4\} \\
&\cup \{f, g, YES, NO\}, \\
\Sigma(\langle n, m\rangle) &= \{s_{i,j}, s'_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}, \\
\mathcal{M}_1 &= YES\ NO\ b_1 c_1 d_1 e_1, \\
\mathcal{M}_2 &= f g a_1 a_2 \ldots a_n, \\
E(\langle n, m\rangle) &= \Gamma(\langle n, m\rangle) - \{YES, NO\}, \\
i_{in} &= 2,
\end{aligned}$$

and the following rules.

1. $[\, a_i \,]_2 \rightarrow [\, T_{i,1} \,]_2\, [\, F_{i,1} \,]_2$, for all $i = 1, 2, \ldots, n$.
2. $(1, b_i / b_{i+1}^2, 0)$, for all $i = 1, 2, \ldots, n+1$.
3. $(1, c_i / c_{i+1}^2, 0)$, for all $i = 1, 2, \ldots, n+1$.
4. $(1, d_i / d_{i+1}^2, 0)$, for all $i = 1, 2, \ldots, n+1$.
5. $(1, e_i / e_{i+1}, 0)$, for all $i = 1, 2, \ldots, 3n + nm + m + 3$.
6. $(1, b_{n+1} c_{n+1} / f, 2)$.
7. $(1, d_{n+1} / g, 2)$.
8. $(2, c_{n+1} T_{i,1} / c_{n+1} T'_{i,1}, 0)$.
9. $(2, c_{n+1} F_{i,1} / c_{n+1} F'_{i,1}, 0)$, for each $i = 1, 2, \ldots, n$.
10. $(2, T'_{i,j} / t_i T'_{i,j+1}, 0)$.
11. $(2, F'_{i,j} / f_i F'_{i,j+1}, 0)$, for each $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$.
12. $(2, b_i / b_{i+1}, 0)$.
13. $(2, d_i / d_{i+1}, 0)$, for all $i = n+1, \ldots, (n+1) + (2n+m) - 1$.
14. $(2, b_{3n+m+1} t_i s_{i,j} / b_{3n+m+1} r_j, 0)$.
15. $(2, b_{3n+m+1} f_i s'_{i,j} / b_{3n+m+1} r_j, 0)$, for all $1 \leq i \leq n$ and $1 \leq j \leq m$.
16. $(2, d_i / d_{i+1}, 0)$, for all $i = 3n+m+1, \ldots, (3n+m+1) + nm - 1$.
17. $(2, d_{3n+nm+m+i} r_i / d_{3n+nm+m+i+1}, 0)$, for all $i = 1, 2, \ldots, m$.
18. $(2, d_{3n+nm+2m+1} / f\ YES, 1)$.
19. $(2, YES / \lambda, 0)$.
20. $(1, e_{3n+nm+2m+2} f\ NO / \lambda, 2)$.
21. $(2, NO / \lambda, 0)$.

6.1 An overview of the computation

Membrane 2 is repeatedly divided, each time expanding one object a_i, corresponding to a variable x_i, into $T_{i,1}$ and $F_{i,1}$, corresponding to the values *true* and *false* which this variable may assume. In this way, in n steps, we get 2^n cells with label 2, each one containing one of the 2^n possible truth assignments for the n variables. The objects f, g are duplicated, hence a copy of each of them will appear in each cell.

In parallel with the operation of dividing cell 2, the counters b_i, c_i, d_i, e_i from cell 1 grow their subscripts. In each step, the number of copies of objects of the first three types is doubled, hence after n steps we get 2^n copies of b_{n+1}, c_{n+1} and d_{n+1}. Objects b_i will check which clauses are satisfied by a given truth assignment, objects c_i are used in order to multiply the number of copies of t_i, f_i as we will see immediately, d_i are used to check whether there is at least one truth assignment which satisfies all clauses, and if such an assignment does not exists, then e_i will be used in order to produce the object NO at the end of the computation.

In step $n+1$, the counters $b_{n+1}, c_{n+1}, d_{n+1}$ are brought in cells with label 2, in exchange of f and g. Because we have 2^n copies of each object of these types and 2^n cells 2, each one containing exactly one copy of f and one of g, due to the maximality of the parallelism of using the rules, each cell 2 gets precisely one copy of each of $b_{n+1}, c_{n+1}, d_{n+1}$. Note that cells 2 cannot divide anymore, because the objects a_i were exhausted.

In the presence of c_{n+1}, the objects $T_{i,1}, F_{i,1}$ get primed, which initiates the possibility of introducing m copies of each t_i and f_i in each cell 2. As we have m clauses, then in order to check their values for a given truth assignment, we need for each clause one set of objects encoding the values of all variables. Note that this phase needs $2n$ steps for priming the objects $T_{i,1}, F_{i,1}$ – for each object we need one step, because we have only one copy of c_{n+1} available – then m further steps for each $T'_{i,1}, F'_{i,1}$; all these steps are done in parallel, but for the last primed $T_{i,1}, F_{i,1}$ we have to continue m steps after the $2n$ necessary for priming. Thus, the total number of steps performed in this process is $2n+m$.

In parallel with the previous operations, the counters b_i and d_i increase their subscripts, until reaching the value $3n+m+1$. This is done in all cells 2 at the same time. Simultaneosly, e_i increases its subscript in cell 1.

In the presence of b_{3n+m+1} – and not before – we check the values assumed by clauses for the truth assignments from each cell 2. We have only one copy of b_{3n+m+1} in each cell, hence we need at most nm steps for this: each clause contains at most n literals, and we have m clauses. In parallel, d increases the subscript, until reaching the value $3n+nm+m+1$.

In each cell with label 2 we check whether or not all clauses are satisfied by the corresponding truth assignment. For each clause which is satisfied, we increase by one the subscript of d, hence the subscript reaches the value $3n+nm+2m+1$ if and only if all clauses are satisfied.

If one of the truth assignments from a cell 2 has satisfied all clauses, then we reach $d_{3n+nm+2m+1}$, which is sent to cell 1 in exchange of the objects YES and f.

In the next step, the object YES leaves the system, signaling the fact that the formula is satisfiable. In cell 1, the counter e will increase one more step its subscript, but after that it will remain unchanged – it can leave cell 1 only in the presence of f, but this object was already moved to cell 2.

If the counter e reaches the subscript $3n+nm+2m+2$ and the object f is still in cell 1, then the object NO can be moved to a cell 2, randomly chosen, and from there it exits the system, signaling that the formula is not satisfiable.

6.2 Some formal details

We consider the polynomial encoding (cod, s) from $I_{\mathtt{SAT}}$ to $\mathbf{\Pi}$, defined as follows: if the formula φ is an instance of `SAT` with size parameters n (number of variables) and m (number of clauses), then

$s(\varphi) = \langle n, m \rangle$ and $cod(\varphi)$ is the set

$$\bigcup_{1 \leq i \leq n, 1 \leq j \leq m, 1 \leq r \leq k_j} \{s_{i,j} \mid y_{j,r} = x_i\} \cup \{s'_{i,j} \mid y_{j,r} = \neg x_i, \}$$

That is, in the multiset $cod(\varphi)$ we replace each variable x_i from each clause C_j with $s_{i,j}$ and each negated variable $\neg x_i$ from each clause C_j with $s'_{i,j}$, then we remove all parentheses and connectives. In this way we pass from φ to $cod(\varphi)$ in a number of steps which is linear with respect to $n \cdot m$.

The presented family of tissue–like recognizer membrane systems is polynomially uniform by Turing machines, because the definition of the family is done in a recursive manner from a given instance of SAT, in particular from the constants n (number of variables) and m (number of clauses). Furthermore the required resources to build the element $\Pi(\langle n, m \rangle)$ of the family are the following:

- Size of the working alphabet: $5nm + 17n + 4m + 12 \in O((max\{n, m\})^2)$.
- Number of membranes: $2 \in \Theta(1)$.
- $|\mathcal{M}_1| + |\mathcal{M}_2| = n + 8 \in \Theta(n)$.
- Maximum length of the rules: 3.

Finally, we can prove, using a formal description of the computations, that the family $\mathbf{\Pi}$ is sound and complete with regard to $(\mathtt{SAT}, cod, s)$. The number of steps of the computations is polynomial in terms of n and m: the answer YES is sent out in step $3n + nm + 2m + 2$, while the answer NO is sent out in step $3n + nm + 2m + 4$.

From the above we deduce the following results:

Theorem 4.

1. $\mathtt{SAT} \in \mathbf{PMC}_{\mathcal{TR}}$.
2. $\mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{TR}}$, *and* $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{\mathcal{TR}}$.

7 Conclusions

In this paper, we have presented cell–like (inspired from the structure of the cell) and tissue–like (inspired from the cell inter–communication in tissues) recognizer membrane systems as computational devices specially suitable to attack the efficient solvability of computationally hard problems.

In that new framework, two characterizations of the relation $\mathbf{P} = \mathbf{NP}$ have been described through the solvability of $\mathbf{NP}$–complete problems by a family of cell–like recognizer membrane systems using only basic rules.

The main contribution of this paper is to present, in the framework of tissue–like recognizer membrane systems, a formal definition of the cellular complexity class $\mathbf{PMC}_{\mathcal{TR}}$.

Recognizer membrane systems with active membranes have been studied in the variants cell–like and tissue–like, and an efficient and uniform solution to the satisfiability problem by tissue–like recognizer membrane systems with cell division has been presented.

Acknowledgement

The authors wish to acknowledge the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds.

References

1. A. Alhazov, C. Martín–Vide, L. Pan. Solving graph problems by P systems with restricted elementary active membranes. In N. Jonoska, Gh. Păun, and G. Rozenberg, editors, *Aspects of Molecular Computing: Essays Dedicated to Tom Head, on the Occasion of His 70th Birthday*, pages 1–22. Lecture Notes in Computer Science, 2950, 2004.
2. A. Alhazov, L. Pan, Gh. Păun. Trading polarizations for labels in P systems with active membranes. *Acta Informaticae*, 41(2-3):111–144, 2004.
3. A. Alhazov and M.J. Pérez–Jiménez. Uniform solution of `QSAT` using polarizationless active membranes. In M.A. Gutiérrez–Naranjo et al., editors, *Proceedings of the Fourth Brainstorming Week on Membrane Computing (vol. I)*, pages 29–40. Report RGNC 02/2006, 2006.
4. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, and A. Riscos–Núñez. A fast P system for finding a balanced 2-partition. *Soft Computing*, 9(9):673–678, September 2005.
5. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, and F.J. Romero–Campero. On the power of dissolution in P systems with active membranes. *Lecture Notes in Computer Science*, 3850:224–240, 2006.
6. T. Head, M. Yamamura, and S. Gal. Aqueous computing: writing on molecules. *Proceedings of the Congress on Evolutionary Computation 1999*, pages 1006–1010. IEEE Service Center, 1999. Piscataway, NJ.
7. L. Pan, A. Alhazov, and T.O. Ishdorj. Further remarks on P systems with active membranes, separation, merging, and release rules. In Gh. Păun et al., editors, *Proceedings of the Second Brainstorming Week on Membrane Computing*, pages 316–324. Report RGNC 01/04, 2004.
8. L. Pan and T.O. Ishdorj. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, 10(5):630–649, 2004.
9. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000, and Turku Center for Computer Science - TUCS Report 208, November 1998, `www.tucs.fi`
10. Gh. Păun. Membrane Computing. An introduction. Springer-Verlag, Berlin, 2002.
11. Gh. Păun, M.J. Pérez–Jiménez, and A. Riscos–Núñez. Tissue P systems with cell division. In Gh. Păun et al., editors, *Proceedings of the Second Brainstorming Week on Membrane Computing*, pages 380–386. Report RGNC 01/04, 2004.
12. M.J. Pérez–Jiménez. An approach to computational complexity in Membrane Computing. In G. Mauri, Gh. Păun, M.J. Pérez–Jiménez, Gr. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC5, Revised Selected and Invited Papers*, pages 85–109. Lecture Notes in Computer Science, 3365, 2005.
13. M.J. Pérez–Jiménez and A. Riscos–Núñez. Solving the Subset-Sum problem by P systems with active membranes. *New Generation Computing*, 23(4):367–384, 2005.
14. M.J. Pérez–Jiménez and A. Riscos–Núñez. A linear-time solution to the knapsack problem using P systems with active membranes. In C. Martín-Vide, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, pages 248–266. Lecture Notes in Computer Science, 2933, 2004.
15. M.J. Pérez–Jiménez and F.J. Romero–Campero. An efficient family of P systems for packing items into bins. *Journal of Universal Computer Science*, 10(5):650–670, 2004.
16. M.J. Pérez–Jiménez and F.J. Romero–Campero. Attacking the Common Algorithmic Problem by recognizer P systems. In M. Margenstern, editor, *Machines, Computations and Universality, MCU'2004*, pages 304–315. Lecture Notes in Computer Science, 3354, 2005.
17. M.J. Pérez–Jiménez and F.J. Romero–Campero. Trading polarizations for bi-stable catalysts in P systems with active membranes. In G. Mauri, Gh. Păun, M.J. Pérez–Jiménez, Gr. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 5th International Workshop, WMC5, Revised Selected and Invited Papers*, pages 373–388. Lecture Notes in Computer Science, 3365, 2005.
18. M.J. Pérez–Jiménez, A. Romero–Jiménez, and F. Sancho–Caparrini. Teoría de la Complejidad en modelos de computación con membranas. Ed. Kronos, Sevilla, 2002.

19. M.J. Pérez–Jiménez, A. Romero–Jiménez, and F. Sancho–Caparrini. Complexity classes in cellular computing with membranes. *Natural Computing*, 2(3):265–285, 2003.
20. M.J. Pérez–Jiménez, A. Romero–Jiménez, and F. Sancho–Caparrini. The P versus NP problem through cellular computing with membranes. In N. Jonoska, Gh. Păun, and G. Rozenberg, editors, *Aspects of Molecular Computing: Essays Dedicated to Tom Head, on the Occasion of His 70th Birthday*, pages 338–352. Lecture Notes in Computer Science, 2950, 2004.
21. A. Riscos–Núñez. *Cellular Programming: efficient resolution of NP–complete numerical problems*. PhD. Thesis, University of Seville, Spain, 2004.
22. A. Romero–Jiménez. *Complexity and Universality in Cellular computing models*. PhD. Thesis, University of Seville, Spain, 2003.
23. C. Zandron, C. Ferreti, and G. Mauri. Solving NP-Complete Problems Using P Systems with Active Membranes. In I. Antoniou, C.S. Calude and M.J. Dinneen, editors, *Unconventional Models of Computation, UMC'2K*, pages 289–301. Springer-Verlag, 2000.

A Dynamic Coalition Formation Approach in the Bicriteria Case

Houda Derbel

Faculté des Sciences Economiques et de Gestion de Sfax, CREM,
Route de l'Aérodrome B.P 1088-3018, Sfax
derbelhouda@yahoo.fr

Abstract. *This paper extends the results regarding coalition formation games and the core to a model of coalition formation in cooperative games based on two criteria. These criteria are money and management. Each criterion is modeled through a suitable characteristic function. Our analysis proceeds at first by establishing the assumptions that guarantee the existence of the core in the bicriteria case. Secondly, by using a dynamic coalition formation algorithm, we developed an algorithm for our model. At each stage, the algorithm provides the player with a better state than the previous one and attains the core if it is possible.*

1 Introduction

Coalition formation models various practical situations as the provision of public goods, the formation of custom union and trading blocs [10]. Coalition formation have an important historical impact [5]. It occurs in our daily life. Several researches were proposed in this framework as [10]. A coalition consists in joining persons to form a single group in order to accomplish one's goal with paying costs as low as possible. Using a game theoretic approach, many people mathematically model the coalition formation in order to determine the winning coalitions that minimize the cost of their activities.

Generally, the money commodity is shown to be a requirement of an economic society. Hence, it allows the formation of coalitions to minimally allocate resources.

In this work, we are dealing with notions and issues of Dynamic Coalition Formation (DCF) among rational agents in a cooperative environment. These agents negotiate to gain and share benefits in stable coalitions by cooperating with other agents in order to minimize costs. The problem of each player is to select the best group that maximizes his profit. This problem calls for the coalition formation as an important solution to find new partners as well as a good method to identify the contribution from each participant.

Generally, the value of a coalition is calculated through a characteristic function v that associates every group with its maximum expected utility obtained when agents join their forces inside the group to achieve their goals. The solution of a coalitional game is a coalition configuration called proposal and is made up of the pair $(\Im, x)$: $\Im$ is a partition of the set of players N, called coalition structure and $x : N \rightarrow IR$ is a payoff distribution where x_i is the payoff of the ith player. A coalition configuration is stable if no agent has a motivation to leave his current coalition to improve his utility.

Among several approaches for searching stability and distributing profits among members of the coalition, we take a particular interest at the core stable coalitions as mentioned in [4, 7, 8]. We are interested in the properties of the core of a coalition formation game evolved when some coalitions appear whereas others disappear. Many algorithms are used to determine the process of coalition formation which purpose is to reach an allocation such as the core [3]. This task requires

the completion of two processes: coalition formation and bargaining about how to split the surplus within each of the coalitions. In general, at each time step, a player decides which of the existing coalitions to join, and demands a share of the payoff which is determined by the characteristic function.

Two related questions are evoked in such problems of coalition formation: Which coalition will form and which are the final designed payoffs to players. In the existing literature, agents devote their efforts to increase their gains, especially monetary gains. Anyway, being the manager in a coalition is an important issue. We think that in a dynamic context, making some disturbance on the distribution of profits may have an effect on the stability of an allocation. Then, we suppose that the utility of one player is assigned according to two criteria: the first is based on money and the second is based on management. Therefore we take an interest in modeling a coalition formation game in which an agent want to receive the higher monetary payoff with being the manager. Our purpose is to answer the two questions listed previously in the case of two criteria through a simple algorithm.

Our work is organized as follow:
In section 2 and 3, we will present the basic concepts and notations in coalition formation games. Besides, we will explain the core, a set of constraints, designed to find the optimal coalition structure. In the next sections, after redefining the concept of the core, we will describe a dynamic coalition formation game with two criteria. Then, we will generate an algorithm for our model which helps a player to reach the most stable coalition structure if there exists to end with an explicit example.

2 Games in Characteristic Function Form (GCF)

In our work, we are interested in cooperative games which are in characteristic function form and have the core as a solution [4].

A characteristic function game is a finite set of players $N = \{1, ..., n\}$ and a real value function v that assigns to every nonempty coalition $S \subset N$ a number $v(S)$ called the worth of S. $v(S)$ denotes the maximum payoff that members of a coalition S can guarantee by cooperation. This concept of cooperation allows the coalition formation in order to formulate rational decision. Coalition formation games enable players to form different coalition structures.
A coalition structure describes the gathering between players in coalitions covered with cooperation.

Definition 1 (Coalition Structure). Let $N = \{1, ..., n\}$ be the set of players participating in a game, then a coalition structure is a finite partition $\Im = \{C_1, ..., C_m\}$ of N such that $\cup_{k=1}^{m} C_k = N$, $C_k \cap C_l = \emptyset$ for $k \neq l$. [1]

3 The Core

The core is a well-known solution, it is the set of feasible utility outcomes giving that no coalition can better the gain of all its members alone. The core generalize the concept of imputations.
Let $x = (x_1, ..., x_n)$ be a payoff distribution, x_i is the utility of doing an activity of the i-th player. It is an imputation if it checks the conditions below:

$$\text{The condition of individual rationality :} \quad v(\{i\}) \leq x_i \tag{1}$$

$$\text{The condition of pareto optimality :} \quad v(N) = \sum_{i \in N} x_i. \tag{2}$$

Analogous to the case of imputation, the core is defined by the two conditions below:

$$\text{The coalitional rationality :} \quad \sum_{i \in S} x_i \geq v(S) \quad \text{for all} \quad S \subset N \tag{3}$$

$$\text{Feasibility :} \quad \sum_{i \in N} x_i = v(N). \tag{4}$$

The core is the set of all feasible allocations that can not be blocked by any coalition. In fact, An allocation x is blocked by a coalition S if $x(S) < v(S)$, that the coalition S can secure more benefits to their members than the proposed allocation.

The core is the set of stable allocations in the sense of equilibrium. Nevertheless, the core solution may be empty. Consequently, a player decides to move from one coalition structure to another in order to maximize his own profit and be in a stable state that can not be blocked by any coalition. To many authors, this decision is taken according to only one criterion: the money. Many authors consider a best coalition only when their members get much more money conditional on feasibility.

A more interesting situation is to take the decision not only to receive much more money but also to be the manager in the coalition structure. In the next section, we will present a new model of a coalition formation game with two criteria: money and management.
We propose to extend the model of [4, 7] to the bicriteria case. A player will prefer a coalition when not only he has a good income relative to others but also being the manager in this coalition. Therefore, the objective of our present work is to develop a methodology to guess which coalitions finally result and offer their members a better position and payoff according to the two criteria.

In the next sections, we present the principle definitions and features of our model as well as the new definition of the core when two criteria are considered: the bicore. Besides, we propose to extend the algorithm of coalition formation analyzed in [7] to our case.

4 The Model

The dynamic nature of the coalition formation process requires that even if a coalition accepts the proposal, it does not stop competing as in static approach but continues bargaining to improve their gains.

Assume that for the transition from one state to another, a player takes into consideration not only the money he will get but also the characteristics of partners in his coalition. Consequently, the decision to join a coalition depends on two complementary criteria: money and management.

Let $N = \{1, ..., n\}$ be the set of players. Consider two characteristic functions v_1 and v_2. The values affected by these two functions are supposed fixed from the beginning.
$v_1 : 2^n \backslash \emptyset \longrightarrow IR$ is the characteristic function that associates every coalition with the maximum quantity of money it can gains from cooperation.
$v_2 : 2^n \backslash \emptyset \longrightarrow \{0, 1\}$ is the characteristic function that indicates which coalitions of parties have the majority.

$$v_2(S) = \begin{array}{l} 1 \text{ if the coalition } S \text{ is in the majority} \\ 0 \text{ otherwise} \end{array}$$

We interpret $v_2(S) = 1$ that the coalition S decides to change its position, it is the leader. Others are just followed it.

We don't consider the case $v_2(\{i\}) = 1$, $\forall i \in N$ because the singleton coalition structure $\{\{1\}, ..., \{n\}\}$ will be formed and the game will be trivial.
We consider two vectors of payoffs: $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$ where X is an imputation defined before and Y is a binary vector. It is a collection of ones and zeros, where "1" is placed opposite a player to indicate that he is selected to be the manager or he is alone and "0" means

that these conditions are not provided.
Let $S(i) = \{S \subseteq N / i \in S\}$ then,

$$y_i = \begin{cases} 1 \text{ if the player } i \text{ is a manager in } S(i) \\ \quad \text{or remains alone} \\ 0 \text{ otherwise} \end{cases}$$

Therefore, a player i would join a player j to form the coalition $\{i, j\}$ if x_i increases and $y_i = 1$ at the next time step. So, a player moves when he is sure to guarantee a better share of profit with the quality of manager.
We assume that $v_2(S)$ and y_i, $i \in S$ are independently affected.

Defining so the second criterion, a realistic situation is that, two managers don't coexist in the same coalition. Hence, $\sum_{i \in S} y_i \leq 1$.
Let our new coalitional game denoted by $\Gamma_{v_1,v_2} = (N, v_1, v_2)$. Then, Given a particular coalition structure $\Im$, an outcome of the coalitional game Γ_{v_1,v_2} called *b*iproposal will be defined by $(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$. The vector X verifies individual rationality and efficiency depending on whether v_1 is given and the vector Y verifies only individual rationality. So as:

$$x_i \geq v_1(\{i\}) \forall i \in N \tag{5}$$

$$\sum_{i \in N} x_i = v_1(\{N\}) \tag{6}$$

$$y_i \geq v_2(\{i\}) \forall i \in N. \tag{7}$$

The third condition captures the fact that any player i tries to keep the managing level or to ameliorate it as he can. We have, as previously defined in the literature, if a group of players can form a coalition which can assure its members a better payoff than the proposed allocation, this coalition will block the allocation. That is, S blocks X if $\sum_{i \in S} x_i < v(S)$.

In our situation, if every player measures his gain according to two criteria, a distribution $\begin{smallmatrix} X \\ Y \end{smallmatrix}$ is blocked if $\sum_{i \in S} x_i < v_1(S)$ for one coalition S in N and at least one player belonging to a leading coalition is a non manager.

Example 1. Consider the coalition game Γ_{v_1,v_2} below:
$N = \{1, 2, 3\}$, $v_1(\{i\}) = 0$, $\forall i \in N$, $v_1(S) = 7$ for $|S| = 2$, $v_1(N) = 12$, $v_2(\{i\}) = 1$, $v_2(S) = 1$, $\forall |S| = 2$, $v_2(N) = 0$.
In the state $(\{N\}, \begin{smallmatrix} 8\ 3\ 2 \\ 1\ 1\ 0 \end{smallmatrix})$, the distribution $\begin{smallmatrix} 8\ 3\ 2 \\ 1\ 1\ 0 \end{smallmatrix}$ is not in the core since it is blocked by the coalition $\{2, 3\}$.
In fact, $8+3+2 = 13 \Rightarrow \sum_{i \in N} x_i \neq v_1(N)$ so one of the feasibility conditions is violated. Moreover, $v_1(\{2, 3\}) = 7 > 3 + 2 = 5$. Since $v_2(\{2, 3\}) = 1$, the coalition $\{2, 3\}$ is one perpetrator and the player 3 is a non manager inside this coalition. So, the coalition $\{2, 3\}$ would deviate to achieve a better position. In conclusion, in the state $(\{N\}, \begin{smallmatrix} 8\ 3\ 2 \\ 1\ 1\ 0 \end{smallmatrix})$, the distribution $\begin{smallmatrix} 8\ 3\ 2 \\ 1\ 1\ 0 \end{smallmatrix}$ is not a stable one.

So, seeing that the core is defined by the set of all feasible distributions that cannot be blocked by any coalition and partition the set of players so that no group wants to form a wew coalition, a state $(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$ belongs to the *bicore* if:

$$\begin{cases} \sum_{i \in S} x_i \geq v_1(S) \ \forall \ S \subseteq N \\ \sum_{i \in N} x_i = v_1(N) \end{cases} \tag{8}$$

$$\begin{cases} \exists S, j \in S / v_2(S) = 0 \text{ and } y_j = 1 \\ \sum_{i \in S} y_i \leq 1 \ \forall \ S \in \Im \end{cases} \tag{9}$$

Denote the bicore by $\mathcal{C}(\Gamma_{v_1,v_2})$.
The equations (8) and (9) involve rationality and feasiblity according to the first and the second criteria respectively.

5 Coalition Structure Generation Algorithm

We will suggest a methodical way for the coalition formation under two criteria. Being in a dynamic context, we propose an algorithm leading either to a solution in the bicore or showing that the bicore is empty.

The algorithm is already used in [7] to show that the core is accessible from any defined state in the one level case. We adapt the steps of the algorithm in our bilevel case.
The coalition formation algorithm is involving four steps that are very related:
In step 1, we specify the set of players that can be condemned for not being in the bicore. We call this set 'overpaid players' [7].
In step 2, we select one deviating coalition.
In step 3, we select a deviating outcome.
In step 4, We repeat step 2 and 3 until we obtain a sequence of dominating biproposals or a biproposal in the core.
We reformulate each step in order to satisfy the assumptions of our model. The algorithm is based on the principle of domination. Firstly, we redefine the domination concept.

5.1 Domination via a Coalition T in the Bicriteria Case

Define the list of vectors designing the payoffs of two players according to the first criterion by $X' = (x'_i)_{i\in N}$, $X = (x_i)_{i\in N}$ and those to the second criterion by $Y' = (y'_i)_{i\in N}$, $Y = (y_i)_{i\in N}$.
We use the domination concept described in [1]. We adapt those definitions when two criteria are considered.

Definition 2 (A loyal group). A coalition L is a loyal group if $\forall I, J \subseteq L$:

1. $v_1(S \cup I) = v_1(S \cup J)$
2. $v_2(S \cup I) = v_2(S \cup J)$
3. $X(I) = X(J)$
4. $Y(I) = Y(J)$

Definition 3 (Domination). A biproposal $(\tau, \begin{smallmatrix} X' \\ Y' \end{smallmatrix})$ dominates a biproposal $(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$ via a coalition $T \in \Im$ iff:

1. $x'_i \geq x_i \ \forall i \in T$ with a strict inequality for at least one player
2. $y'_i \geq y_i \ \forall i \in T$ with a strict inequality for at least one player
3. τ contains all coalitions of $\Im$ that don't intersect T
4. If there exists a loyal group $L \in T$ then $L \in \tau$

We note $(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix}) \longrightarrow_T (\tau, \begin{smallmatrix} X' \\ Y' \end{smallmatrix})$

5.2 Notations for the Algorithm

In this section, we give all the notations used in the algorithm analysed in section 5.3.
N: The set of players
x_i: The payoff of the player i in money
$X = (x_i)_{i\in N}$: The vector of payoffs according to the first criterion: money

y_i: $y_i = 1$ if the player i is a manager and $y_i = 0$ if he is a non manager
$Y = (y_i)_{i \in N}$: The vector of payoffs according to the second criterion: managing
$X(S) = \sum_{i \in S} x_i$: The ex-post value of the coalition S according to the first criterion
$Y(S) = \sum_{i \in S} y_i$: The ex-post value of the coalition S according to the second criterion
$v_1(S)$: The ex-ante value of the coalition S according to the first criterion
$v_2(S)$: The ex-ante value of the coalition S according to the second criterion
L: A loyal group
$\begin{smallmatrix} X \\ Y \end{smallmatrix}$: The matrix of ex-post payoffs
$\Im$: A coalition structure
$(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$: A biproposal or a state of the game
$(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix}) \longrightarrow_T (\tau, \begin{smallmatrix} X' \\ Y' \end{smallmatrix})$: The biproposal $(\tau, \begin{smallmatrix} X' \\ Y' \end{smallmatrix})$ dominates the biproposal $(\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$ via the coalition $T \in \Im$
$O(b, a)$: Overpaid players relative to a
$|C \cap S|$: The number of players in $C \cap S$
$|.|$: The absolute value
$s_i \hookrightarrow_C s_j$: The coalition C deviates from the state s_i to give rise to the state s_j.

5.3 Analyzing the Different Steps of the Algorithm in the Bicriteria Case

Step 1: Selecting overpaid players.
During this step, we determine the set of overpaid players, those that can't help for going to the bicore and we can't sentence them for this behavior.
Let $b_0 = (\Im_0, \begin{smallmatrix} X_0 \\ Y_0 \end{smallmatrix})$ be the initial state that does not belong to the bicore and $a = (\Im, \begin{smallmatrix} X \\ Y \end{smallmatrix})$ be an element of the bicore. A player i such that $x_{0i} > x_i$ and $y_{0i} \geq y_i$ is said to be overpaid relative to a. We denote the set of overpaid players by $O(b_0, a)$.

If a dominates b_0 via a particular coalition, the process is almost finished. If it is the case, the latter will be a stable state of the coalition formation process.
Otherwise, we look for a state $a^* = (\Im^*, \begin{smallmatrix} X^* \\ Y^* \end{smallmatrix})$ in the bicore that minimizes the number of overpaid players. We call a^* an efficient core state.

Therefore, we want to minimize $|X_0(S) - X^*(S)| = |\sum_{i \in S} x_i - \sum_{i \in S} x_i^*|$ and makes $y_{0i} - y_i^* = 0$ or $|Y_0(S) - Y^*(S)| = |\sum_{i \in S} y_i - \sum_{i \in S} y_i^*|$ as minimum as possible for players in the overpaid set relative to a^* denoted by $S = O(b_0, a^*)$.
Here, if the number of bicore states is more than two, we keep only the two states that assures the minimum surplus according to the first criterion.We suppose that players in these cases favor the money to the management.
Step 2: Selecting a deviating coalition C.
In this step, we look for blocking coalitions. Indeed, a coalition C blocks the outcome $\begin{smallmatrix} X_0 \\ Y_0 \end{smallmatrix}$ if: $v_1(C) > X_0(C)$ and $v_2(C) = 1$ and $y_{0i} = 0 \forall i \in C$.
We begin by searching such coalitions in $\Im^*$. If $\Im^*$ does not contain a blocking coalition, we choose a minimal existing one. If there does not exist such coalition, the player with the worst position according to the first criterion decides to deviate. Otherwise, the game still static and players can't move from one state to another. When a coalition C is deviates to reach a new state b_1, we note $b_0 \hookrightarrow_C b_1$.
Step 3: Selecting a deviating outcome.

We would fix a new outcome $b_1 = (\Im_1', \begin{smallmatrix} X_1' \\ Y_1' \end{smallmatrix})$ with reference to the deviating coalition C. In this stage, we have $X_0(C) < v_1(C) \leq X^*(C)$, $v_2(C) = 1$ and $y_{0i} = 0 \ \forall i \in C$, $X^*(N) = v_1(N)$ and $\exists S, j \in S$ s.t $v_2(S) = 0$.
After one possible deviation, overpaid players in the deviating coalition divide the surplus fairly.

Inside the deviating coalition, the player who has the maximum value according to the first criterion becomes a manager. Hence, the new values are:

$$x_{1i}^{'} = x_{0i} + \frac{1}{|C \cap S|}[v_1(C) - X_0(C)] \quad \text{if} \quad i \in O(b_0, a^*) \cap C \tag{10}$$

$$x_{1i}^{'} = x_{0i} \quad \text{if} \quad i \in C - O(b_0, a^*) \tag{11}$$

$$y_{1i}^{'} = \begin{cases} 1 & \text{if } x_{0i} = max_{j \in C} x_{0j} \\ y_{0i} & \text{otherwise} \end{cases} \tag{12}$$

The ex-partners of C become singletons and receive their reservation payoffs: $(v_1(\{i\}), 1)$.
Step 4: If b_1 belongs to the bicore or b_1 is dominated by an element of the bicore, stop. Otherwise, repeat step1, 2 and step 3 with $b_0 = b_1$.

5.4 Example of an Application of the Algorithm to the Case of Three Players

In the coalition game Γ_{v_1,v_2} described before, we distinguish two computational problems. First, we have to find all coalitions and the values of those associated to the criterion. If we assume that the set of players is $N = \{1, ..., n\}$, then there are $2^n - 1$ coalitions. Second, we will specify different coalition structures which grow when the number of players grow. This number is $\sum_i^n S(n, i)$ where $S(n, i)$ is the coalition structure composed with i coalitions known as the Stirling number catched in a recurrent way:

$$\begin{cases} S(n, n) = S(n, 1) = 1 \\ S(n, i) = S(n-1, i-1) + iS(n-1, i) \end{cases} \tag{13}$$

Proposition 1. *The number of coalition structures is $O(n^n)$ [11].*

The number of coalition structures is so big to have an accurate optimal solution. In our application, the problem is not ours. We are interested with only three players. So, we deal only with seven coalition and five coalition structures.

Example 2. Let $N = \{1, 2, 3\}$ be the set of players. Different values of coalitions under the two characteristic functions are as follow:
$v_1(\{1\}) = 1$, $v_1(\{2\}) = 1$, $v_1(\{3\}) = 2$, $v_1(\{1, 3\}) = 7$, $v_1(\{1, 2\}) = 0$,
$v_1(\{2, 3\}) = 4$, $v_1(\{1, 2, 3\}) = 8$
$v_2(\{1\}) = 1$, $v_2(\{2\}) = 1$, $v_2(\{3\}) = 1$, $v_2(\{1, 3\}) = 1$, $v_2(\{1, 2\}) = 0$,
$v_2(\{2, 3\}) = 0$, $v_2(\{1, 2, 3\}) = 1$.
There are five states, each one is described by a coalition structure and a distribution of payoffs according to the two criteria. We denote a state s_i by the pair $(\Im_i, \begin{smallmatrix} X_i \\ Y_i \end{smallmatrix})$, $X_i = (x_{i1}, x_{i2}, x_{i3})$ and $Y_i = (y_{i1}, y_{i2}, y_{i3})$.

$s_0 = (\{\{1\}, \{2\}, \{3\}\}, \begin{smallmatrix} 3\ 1\ 7 \\ 1\ 0\ 0 \end{smallmatrix})$,

$s_1 = (\{\{1, 2\}, \{3\}\}, \begin{smallmatrix} 1\ 1\ 6 \\ 1\ 0\ 0 \end{smallmatrix})$,

$s_2 = (\{\{1, 3\}, \{2\}\}, \begin{smallmatrix} 3\ 1\ 4 \\ 1\ 0\ 0 \end{smallmatrix})$,

$s_3 = (\{\{1\}, \{2, 3\}\}, \begin{smallmatrix} 2\ 2\ 2 \\ 1\ 1\ 1 \end{smallmatrix})$,

$s_4 = (\{N\}, \begin{matrix} 4\ 2\ 2 \\ 1\ 0\ 0 \end{matrix})$.
$E_0 = \{s_0, s_1, , s_2, s_3, s_4\}$.
We suppose that these states are a priori states. Through the algorithm, they change into posteriori states.
The bicore is $\mathcal{C}(\Gamma_{v_1,v_2}) = \{(\Im, \begin{matrix} x_1\ x_2\ x_3 \\ y_1\ y_2\ y_3 \end{matrix})$ such that $x_1 + x_2 + x_3 = 8$,
$X(S) = \sum_{i \in S} x_i \geq v_1(S), \forall S \subseteq N$ and $\exists S \subseteq N \,/\, v_2(S) = 0$ and
$y_j = 1, j \in S$ and $\sum_{i \in S} y_i \leq 1 \ \forall S \in \Im\}$.
First, we determine states which are in the bicore. We remark that
$\mathcal{C}(\Gamma_{v_1,v_2}) = \{s_1, s_2\}$. Generally we start our algorithm with a singleton coalition structure which is a non bicore state, otherwise the coalition formation process is inessential.

Stage 1

Step 1
Let $b_0 := s_0$ and $a := s_1$. There is no coalition T in $\Im_0$ such that s_1 dominates s_0 via T.
The set of overpaid players relative to s_1 is :
$O(s_0, s_1) = \{i \in N \,/\, x_{0i} > x_{1i}$ and $y_{0i} \geq y_{1i}\}$, the set of players in $\Im_0$ that have much more money with at least the same level of managing in the coalition structure $\Im_1$. After a few calculation, we conclude that $O(s_0, s_1) = \{3\}$. In this case, we look for one state a^* belonging to the bicore $\mathcal{C}(\Gamma_{v_1,v_2})$ that minimizes $X_0(\{3\}) - X_i(\{3\})$ and $Y_0(\{3\}) - Y_i(\{3\})$ for $i \in \{1, 2\}$. We have :
$X_0(\{3\}) = x_{03} = 7$, $Y_0(\{3\}) = y_{03} = 0$,
$X_1(\{3\}) = x_{13} = 6$, $Y_1(\{3\}) = y_{13} = 0$,
$X_2(\{3\}) = x_{23} = 4$, $Y_2(\{3\}) = y_{23} = 0$,
So, the efficient core state is $a^* = s_1$.
Step 2
We look first for a deviating coalition in the partition $\Im_1 = \{\{1, 2\}, \{3\}\}$ corresponding to the coalition structure of a^*. We do find a coalition S that blocks s_0, that is to verify these two conditions:

1. $v_1(S) > X_0(S)$
2. $v_2(S) = 1$ and $y_{0j} = 0$, for at least one $j \in S$

We remark that none of the coalitions verify these conditions. Hence, we choose a player who is in a worst position. The payoffs of the singleton player 2 are the worst. Hence, the deviating coalition is $C = \{2\}$.
Let the new state be $b_1 = (\Im_1^{'}, \begin{matrix} X_1^{'} \\ Y_1^{'} \end{matrix})$, we conclude that $s_0 \hookrightarrow_{\{2\}} b_1$.
Step 3
We have to select a deviating outcome as shown previously: the players in $C = \{2\}$ that are overpaid relative to s_1 receive their initial value with dividing up the surplus: $v_1(\{2\}) - X_0(\{2\})$. Those in C and not overpaid keep their initial value. Moreover, ex-partners of C receive their reservation value $(v_1\{i\}, 1)$.
The set of overpaid players relative to s_1 is: $O(s_0, s_1) = \{3\}$
$C \cap O(s_0, s_1) = \emptyset$. So,
$x_{12}^{'} = x_{02} = 1$, $y_{12}^{'} = 1$,
$x_{11}^{'} = x_{01} = 3$, $y_{11}^{'} = y_{01} = 1$
$x_{13} = x_{03} = 7$, $y_{13} = y_{01} = 0$.
Finally $b_1 = (\{\{1\}, \{2\}, \{3\}\}, \begin{matrix} 3\ 1\ 7 \\ 1\ 1\ 0 \end{matrix})$
Step 4
If the new state is in the bicore or dominated by a state in the bicore, the process is finished and

this last state is a stable one. First, b_1 is not in the bicore because $X_1^{'}(\{1,2,3\}) = 3+1+7 = 10 \neq v_1(\{1,2,3\}) = 8$. Second, s_2 and s_1 don't dominate b_1. So, we repeat the same steps with a new initial state equal to b_1.
At last, we obtain a sequence of deviating biproposals: $s_0 \hookrightarrow_{\{2\}} b_1$.
We note that in this case, at the next stages, the new state remains the same and is equal to b_1 which is not in the bicore. Therefore, we can't reach a bicore state for this game.

Contrary to the model of [7] that shows that the core is accessible for only one criterion, this study should be extended to adress several issues. However, it takes into consideration the management issue which is an important and an advantageous situation in some society.

6 Conclusion and Future Research

In this paper, we described a new cooperative game among players measuring their payoffs according to two criteria: money and managing. For this purpose, we extended the coalition game suggested in [4] with only one characteristic function to a game with two characteristic functions.
After that, we paid our attention to the core. We gave a suitable definition of the core which reflects stable states under the two criteria.
We developed an algorithm basically founded from our definitions of the bicore concept and blocking coalitions, those that inhibit or permit a state to be in the bicore. This algorithm permits us to give an accurate solution for a player in order to help him to be in the most stable coalition structure. We notice that there are games where the algorithm falls in a case with cyclical steps as shown in the last example. In this case, the game can't have a realistic stable state.
In future research, we firstly intend to enlarge the size of the games that were relatively small. Secondly, we should formulate a coalition formation problem with muliticretria and define the equilibrium in this case.

Acknowledgments:

The author is grateful to Professor Foued BEN ABDELAZIZ, Associate Professor at the Management University of Sharjah, for his support and constructive criticisms.

References

1. Albers, Schulz and Mueller. Social dilemmas and cooperation. *Springer-Verlag*, 1994.
2. K. Bernhard and J. Rosennüler. Modern applied mathematics optimization and operations research. *N.H.P.C: North Holland Publishing Company*, 1982.
3. J. Contreras, M. Klush and J. Yen. Multi-agent coalition formation in power transmission planning: a bilateral shapley value approach. *American Association for Artificial Intelligence*, 1998.
4. T. Dieckmann and U. Schwalbe. Dynamic coalition formation and the core. *Economics department working paper, Series n810798, departement of economics national university of Ireland, Maynooth*, 1998.
5. W.A. Gamson. A theory of coalition formation. *American Sociological Review*, (21), 1961.
6. M. Klush and O. Shehory. A polynomial kernel-oriented coalition formation algorithm for rational information agents. *Multi-agent Systems, AAAI Press*, pages 157–164, 1996.
7. L.Á. Kóczy and L. Lauwers. The coalition structure core is accessible. *Games and economic behavior*, elsevier, (48), pages 86–93, 2004.
8. H. Konishi and R. Debraj. Coalition formation as a dynamic process. *Journal of Economic Theory*, (110), pages 1-41, 2003.

9. D. Ray and R. Vohra. Equilibrium binding agreements. *Journal of Economic theory*, (73), pages 30–78, 1997.
10. D. Ray and R. Vohra. Coalitional power and public goods. *Journal of Economics*, (109), No. 6, 2001.
11. T. Sandholm, K. Larson, M. Andersson, O. Shehory and F. Tohmé. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, (111), pages 209-238, 1999.

Growth patterns applied to structural mapping

Rafael Jurado Piña, Luisa María Gil Martín, and Enrique Hernández-Montes

Department of Structural Mechanics
University of Granada
Campus de Fuentenueva
18072 Granada
Spain
emontes@ugr.es

Abstract. *Most of the present structures in the nature are structures of uniform tension. There are a multitude of examples: branches of trees, the tubular form of the alive beings, etc. The simplest example can be a soap bubble. One of the great problems to which engineers have to afford to calculate a structure is its discretization. Discretization is necessary in order to apply structural analysis methods as MEF or BEM and calculate displacements and deformations of the structure. In this article we present a new mapping method inspired by the growth of the alive beings instead of in the final form of the structure so and as it has been made until now. This new method is giving better results than its predecessors in some problems of engineering, as they are the tightened structures or the pneumatic structures. The most important contribution of this new approach is that due to mapping in topology an initial guess of the equilibrium position is not needed as part of the solution process. The easiness of implementation is also illustrated in this paper, along with future research that may improve this new idea. Several examples are presented that illustrate the applicability of this new method.*

1 Introduction

The great difference between structures done by the man, like the buildings, and the structures of the nature, like the trees, are that first they become as they are conceived and the last ones grow. Often the engineers we are ourselves forced to construct such and as it makes the nature and our initial intuition is no longer valid. For example, we imagine that we want to construct an inflatable cover with several conditions of contour; most healthful it is to make a bubble of soap and to copy its form, see Figure 1. If we try to construct something like that we will have serious problems of engineering if we did not copy the soap bubble as the famous German engineer Frei Otto did.

In this paper we displayed a discretization based on the growth and not on the final form. From this point of view, topologic patterns are more adapted than the geometric ones. The formers reflect a growth prototype. In this publication we have applied this idea to a simple problem: the existing balance in the structures of constant stress as they can be the soap bubbles.

Topological techniques based on growth patterns have been presented recently by Hernández-Montes et al (2006).

Equilibrium in constant stress structures

The force density method (FDM) was initially presented by Linkwith and Shek (1971). This method is based upon the force-length ratios or force densities which are defined for each branch of the net structure. Shek (1974) shows that the force densities are very suitable for the description of

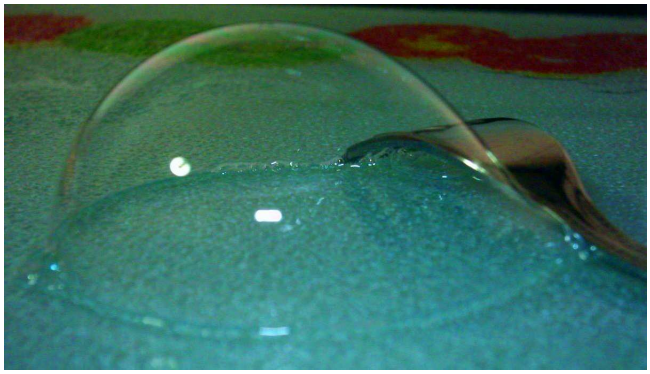


Fig. 1. Shape adaptation of a soap bubble.

the equilibrium state of any general network. The FDM renders a simple linear system of equations for a possible initial configuration. FDM was a big improvement over the Grid Method (Siev 1964) which is one of the most simple methods of form finding. In the Grid Method a horizontal grid is laid out under the structure and horizontal displacements of the joints are neglected, so the equilibrium is solved in the horizontal plane. Vertical equilibrium for each of the joints is used to calculate the elevation of each of the grid points through a linear system of equations.

In the FDM equilibrium shape and prestress distribution are unknowns. Haber and Abel (1982a and 1982b) presented the "smoothing concept" which allows the designer to solve for the shape of the reference configuration in terms of an assumed prestress distribution. Using this method the equilibrium equations are nonlinear, and consequently iterative techniques are required for the solution. A good example of today's use of linear methods can be seen in Levy and Spillers (1998). They use FDM and the Grid-Method as a simple practical approach to design through the use of geometrically nonlinear analysis.

2 The force-density method

The starting point for the FDM is a pin-joint network consisting of cable or bar elements, in which some of the points are fixed and the others are free, and the free points will have to find a position in the equilibrium configuration. This paper is related to this starting point, and the objective is to generate the pin-joint network topologically rather than geometrically, as done so far.

As we can contemplate through the many examples presented, for example, in the book of Levy (2004), an initial guess of the shape is needed in order to generate a network, for instant the approximate shape of the horizontal projection. With this initial shape it is easy to consider a network even when we pick only the connections as input in the FDM, this way of doing is called "geometrical mapping" in this paper, in order to differentiate it from the method proposed here.

For the sake of simplicity the notation used by Shek (1974) for the force density method is also used here. Small changes and explanations are also introduced in order to make the paper self-contained and in order to apply the use of the topological mapping proposed in this paper.

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix}$$

For a given pin-joint network with n nodes and m branches, the branch-node matrix $\mathbf{C}$ is a $m \times n$ matrix used in the FDM to define the connectivity of the nodes as indicated in Figure 2. As

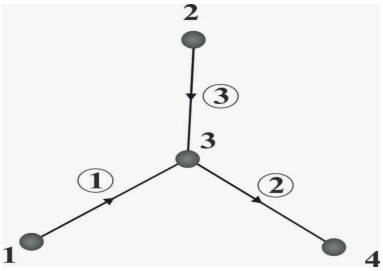


Fig. 2. Example for the construction of the branch-node matrix.

shown in Figure 2, each branch or connection j links two nodes $i(j)$ and $k(j)$,for i<k the elements of the branch-node matrix $\boldsymbol{C}$ can be defined as follows:

$$\mathrm{C}(j,r) = \begin{cases} +1 & if\ i(j) = r \\ -1 & if\ k(j) = r \\ 0 & for\ the\ rest \end{cases} \tag{1}$$

The nodes P_i have coordinates (x_i, y_i, z_i), *i=1, ..., n.* Some of these nodes are fixed, and they will constitute the input data for the initial equilibrium configuration problem. The x, y and z-coordinates for each of the nodes can be grouped in the $n-$vectors $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$. The m-vector $\boldsymbol{l}$ contains the lengths l_j of each branch j, and the vector $\boldsymbol{s}$ is the $m-$vector formed by the branch forces s_j of each branch j. The nodal loads are characterized by means of n-vectors $\boldsymbol{p}$ containing the force components $\boldsymbol{p}_x$, $\boldsymbol{p}_y$ and $\boldsymbol{p}_z$.

The equilibrium equations for each node may now be established. In the case of Figure 2, the equilibrium equations for node 3 come from the projections of the forces along each one of the axes:

$$\begin{array}{l} \frac{s_1}{l_1}(x_3 - x_1) + \frac{s_2}{l_2}(x_3 - x_4) + \frac{s_3}{l_3}(x_3 - x_2) = p_{3x} \\ \frac{s_1}{l_1}(y_3 - y_1) + \frac{s_2}{l_2}(y_3 - y_4) + \frac{s_3}{l_3}(y_3 - y_2) = p_{3y} \\ \frac{s_1}{l_1}(z_3 - z_1) + \frac{s_2}{l_2}(z_3 - z_4) + \frac{s_3}{l_3}(z_3 - z_2) = p_{3z} \end{array} \tag{2}$$

The great advantage of the FDM is the introduction of a parameter q, which is defined as the force-length ratio or force density for the branches. If q is constant then the equilibrium equations become a linear system.

For a more general formulation of the FDM, the coordinate differences $\boldsymbol{u}$, $\boldsymbol{v}$ and $\boldsymbol{w}$ of the connected nodes are considered. These are m-vectors that can be obtained using the $\mathbf{C}$ matrix as follows:

$$\begin{array}{l} \mathbf{u} = \mathbf{Cx} \\ \mathbf{v} = \mathbf{Cy} \\ \mathbf{w} = \mathbf{Cz} \end{array} \tag{3}$$

Considering that the matrices $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$ and $\boldsymbol{L}$ are the diagonal matrices resulting from placing the vectors $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$ and $\boldsymbol{l}$ in the main diagonal, the equilibrium equations for the complete network may be cast in the following form:

$$\begin{array}{l} (\mathbf{C^t U L^{-1}})\mathbf{s} + \mathbf{p_x} = \mathbf{0} \\ (\mathbf{C^t V L^{-1}})\mathbf{s} + \mathbf{p_y} = \mathbf{0} \\ (\mathbf{C^t W L^{-1}})\mathbf{s} + \mathbf{p_z} = \mathbf{0} \end{array} \tag{4}$$

A key property of the FDM is that the above system of equations is a linear system under the following assumption:

$$\mathbf{q} = \mathbf{L}^{-1}\mathbf{s} \tag{5}$$

where $\mathbf{q}$ is an m-vector containing the force-density ratios q_j of all the branches. Finally, taking into account the following identities:

$$\begin{aligned}(\mathbf{C^T Q C})\mathbf{x} + \mathbf{p_x} &= \mathbf{0} \\ (\mathbf{C^T Q C})\mathbf{y} + \mathbf{p_y} &= \mathbf{0} \\ (\mathbf{C^T Q C})\mathbf{z} + \mathbf{p_z} &= \mathbf{0}\end{aligned} \tag{6}$$

And letting $\boldsymbol{Q}$ be the diagonal matrix that contains $\boldsymbol{q}$, the following equilibrium equations may be obtained:

$$\begin{aligned}\mathbf{Uq} &= \mathbf{Qu} \\ \mathbf{Vq} &= \mathbf{Qv} \\ \mathbf{Wq} &= \mathbf{Qw}\end{aligned} \tag{7}$$

The above system of equations constitutes a linear system where the known values are the coordinates of the fixed points, the topology of the pin-joint network, and the force density values. The unknowns are the coordinates of non-fixed nodes.

3 Growth Patterns

Topology mapping is based on the idea of different growth patterns together with the fact that no initial coordinates are needed for the FDM, rather the connectivity between nodes. Taking advantage of this property a mapping is performed in topology, so the initial guess of the shape is no longer needed. This procedure has been called "topological mapping" (Hernández-Montes et al. 2006)

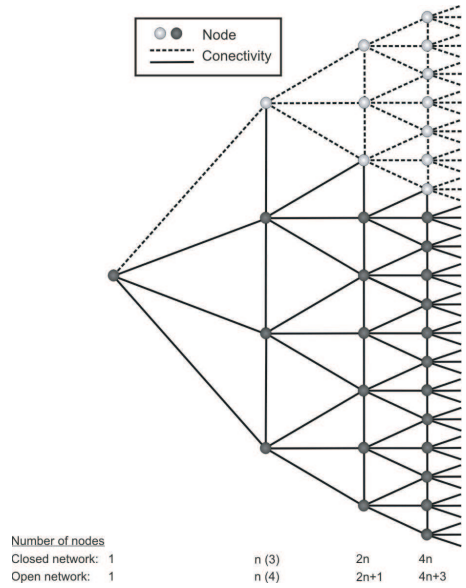
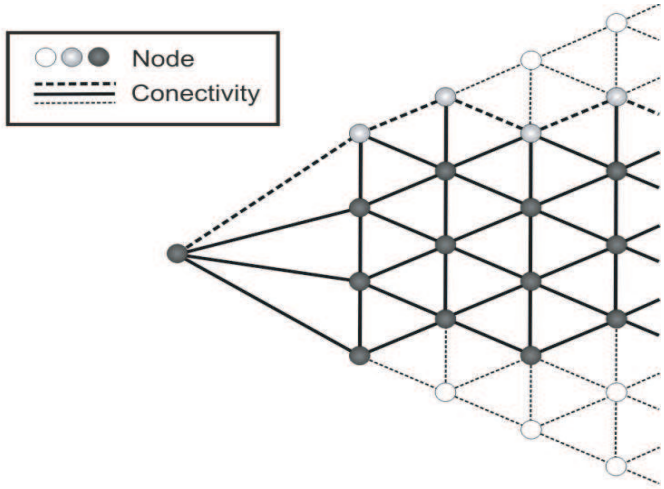


Fig. 3. Growth pattern or basic network type A.

The main feature of the topological mapping is that with a few topological rules a mapping can be performed independently of the final geometric configuration. Based on this new concept

many types of topological networks can be selected and performed. Three **basic growth patterns** are presented here: type A, type B and type C that correspond to Figures 3, 4 and 5, respectively. The basic networks are classified in close network (for close structures a cells or animal bodies) and open network as tress.

In case of the type A open basic network (Figure 3), the connectivity between nodes is represented by thick and dashed lines, and the nodes are black or gray (with no difference among them). In case of the type A close basic network (Figure 3), the connectivity between nodes is represented by thick lines, and dashed lines mean repeated connections. In the case of the type A close basic network the black circles represent nodes, and the gray circles represent repeated nodes.



Number of nodes					
Closed network:	1	n (3)	n	n	n
Open network:	1	n (4)	n+1	n+2	n+3

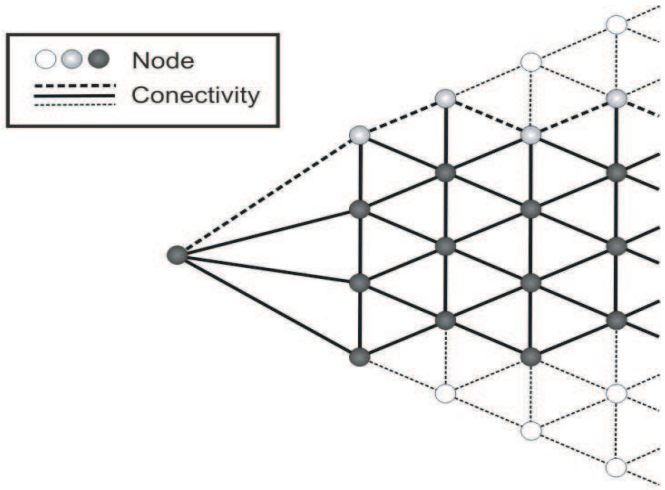
Fig. 4. Growth pattern or basic network type B.

Three symbols are needed in order to illustrate growth pattern type B (Figure 4). Black, gray and empty circles mean nodes in the open network or growth pattern type B. In the case of close networks, a black circle means node, gray circle means repeated node and empty circle means non existing node. The same for the thick line, dashed thick line and dashed thin line, all of them mean connectivity in open network; and for close network they mean connectivity, repeated connectivity and non existing connectivity, respectively.

The behavior of the basic network type C (Figure 5) open or close is similar to network type A.

The basic network A correspond to a pattern in which each node at a given step is connected to the adjacent ones on the same step as well as three more nodes of the following step. In this way the number of nodes of each step is double the amount of the previous one for the closed configuration, and double plus one node for the open configuration.

The basic network B is similar to type A with the difference that each node at a given step is connected to two nodes of the following step. With network B the number of nodes of each step is equal to that of the previous step for the closed configuration and the previous plus one for the open configuration.



Number of nodes					
Closed network:	1	n (3)	n	n	n
Open network:	1	n (4)	n+1	n+2	n+3

Fig. 5. Growth pattern or basic network type C.

The pattern of basic network C is such that each node at a given step is connected alternatively to one or three nodes of the next step. In this way the number of nodes in each step remains the same for both open as well as closed networks.

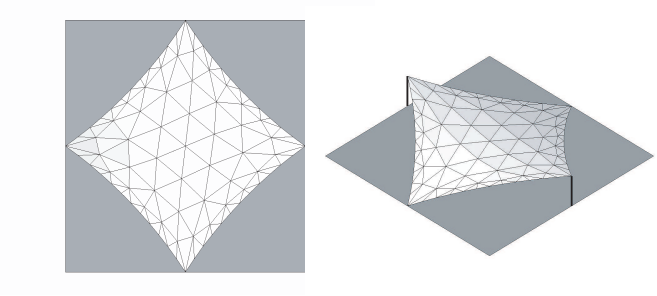
The described basic networks originate from an initial node that is connected to the n nodes of the second step. However, this initial node may be suppressed in open networks, as well as closed networks whenever it is desired to create a new interior contour in the interior of the net.

The algorithms that generate close networks require a reduced and simple data input, namely: the location of the fixed nodes, the force densities of the interior and exterior branches, and the topology of the net. The topology is characterized by the number of nodes of the second step, the number of steps and the basic pattern. The distribution of nodes of the last step along the contour as well as the fixed points is done in an automatic way for closed networks. In this way for any given contour, it is possible to directly generate an equilibrium configuration. On the other hand, with open network is more complicated since all the steps share nodes on the given contour.

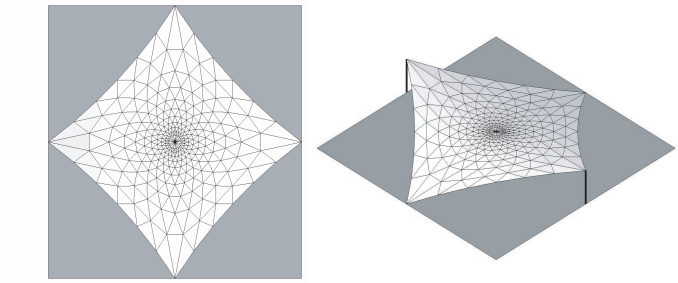
An example of the solutions of the form-finding problem solved using the network types A, B and C are illustrated in Figures 6a, 6b and 6c for closed basic networks. The solutions for the case of open networks are shown in Figures 7a, 7b and 7c. For the closed networks the nodes in the last step are the nodes located at the exterior of the final configuration as well as the fixed nodes. The amount of nodes between consecutive fixed points is proportional to the length between them. In the case of open networks the outside nodes correspond to the outside of the graph tree.

4 Simple networks

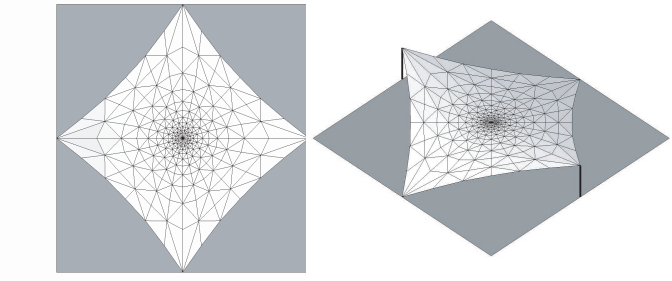
The basic networks may present several problems if directly applied. For example, types B and C require a high number of nodes in their second step for closed networks. On the other hand, network A produces a high number of nodes just after a few steps. These limitations may be eliminated by using topology combinations of the basic networks A, B and C to form what we call **simple**



(a) Closed basic network type A: perspective and upper view.



(b) Closed basic network type B: perspective and upper view.



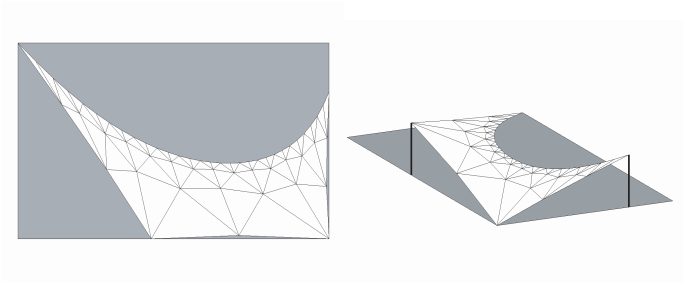
(c) Closed basic network type C: perspective and upper view.

Fig. 6. Example of closed basic network type A, B and C.

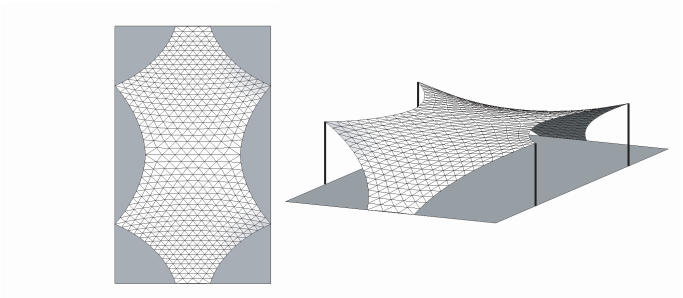
networks. In this case, the user needs to define the desired sequence of combinations. This type of network becomes specially suited whenever the final configuration has a sensible radial symmetry. The use of simple open networks is especially indicated for the case of elongated equilibrium configurations as illustrated in Figure 8.

For practical application of this procedure, the fixed points of the structure correspond to nodes of the growth pattern. The allocation of these nodes is by means of numerical algorithms that consider the distances between these fixed points. The number of steps in each growth pattern is selected by the engineer. The number of steps will be chosen that produces the wished resolution.

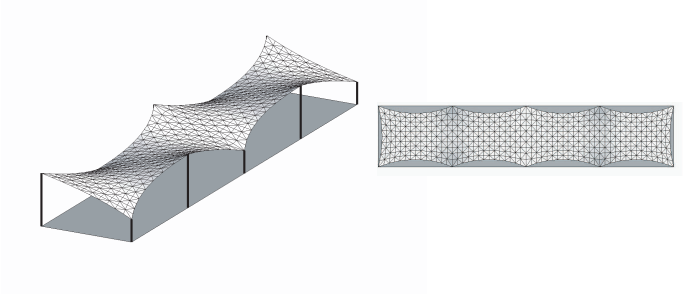
Recently an experiment has been carried out applying the techniques of discretization based in growth patterns. The experiment has supposed the first transmission of technology to the industry of this new technique, Figure 9.



(a) Open basic network type A: perspective and upper view.



(b) Open basic network type B: perspective and upper view.



(c) Open basic network type C: perspective and upper view.

Fig. 7. Example of open basic network type A, B and C.

5 Conclusions

In this paper a procedure based on growth patterns for mapping is presented. The method deals with several growth patterns and it is topology based. The method is easy to use and leads to different choices of final configurations. As first step these methods have been applied to structural simple behaviors and constitute a straightforward and efficient method to obtain a first equilibrium configuration of any type of tension structures.

6 Acknowledgments

We extend our sincere appreciation to the Applied Mathematics Department of the University of Granada and specially to Professors Miguel Pasadas and Pedro Torres for their stimulating discussions on graph theory.

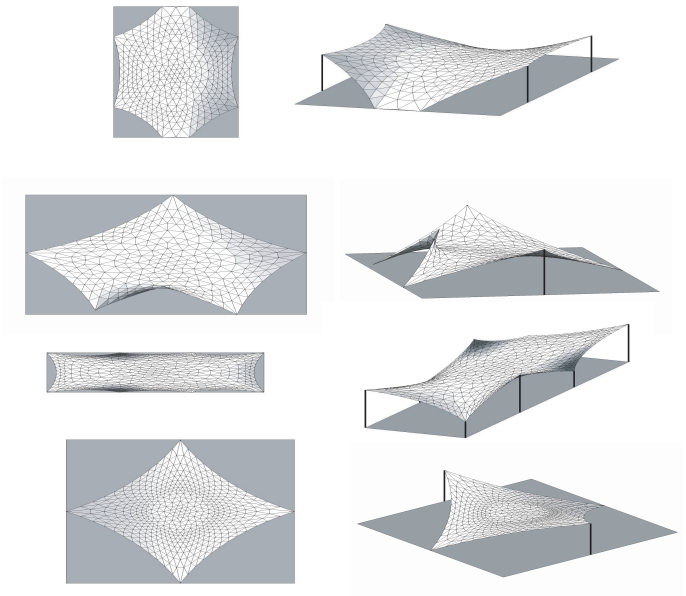


Fig. 8. Examples of the adaptability of simple closed networks to the different contour conditions.

Fig. 9. Experiment of discretization based on Growth patterns.

References

Frei Otto. See at www.freiotto.com

Siev, A., and Eidelman, J. (1964). "Stress analysis of prestressed suspended suspended roofs", *Proc. ASCE*, 90, ST4, 103-121, August.

Schek, H.J. (1974). "The force density method for form finding and computation of general networks," *Computer Methods in Applied Mechanics and Engineering*, 3, 115-134.

Hernández-Montes, E. Jurado-Piña R. and Bayo E.(2006) "Topological mapping for Tension Structures". *ASCE Journal of Structural Engineering.* Volume 132, No.6. June. 970-977.

Levy, R., and Spillers, W.R. (1998). "Practical method of shape finding for membranes and cable nets," *Journal of Structural Engineering,*124 (4), 466-468.

Levy, R., and Spillers, W.R. (2004) *Analysis of Geometrically Nonlinear Structures.* Second Edition, 2004. 288 p. Ed. Chapman & Hall.

Linkwitz K., and Shek H.J. (1971). "Einige Bemerkung von vorsgepannten Seilnetzkonstruktionen". Ingenieur-Archv 40, Springer Verlag, 1971, 145-158.

Barnes, M.R. (1999). "Form finding and analysis of tension structures by dynamic relaxation," *International Journal of Space Structures,* 14(2), 89-104.

Appendix. Notation

$C = branch - nodematrix$

$i(j)$= node i of branch j

j= branch

$k(j)$= node k of branch j

l_j =length of branch$j$

$\boldsymbol{l}$ =vector of the length of the branches

$\boldsymbol{p}_x$, $\boldsymbol{p}_y$,$\boldsymbol{p}_z$= vectors of the force components for each node

q, q_j= force-density ratio of branch j.

$\boldsymbol{q}$ = vector containing the force-density ratios q_j of all the branches

r,P_i =node

s_j= force of the branch j

$\boldsymbol{s}$ =vector formed by the branch forces

u, v, w=coordinate difference of the connected nodes

$\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{L}$,$\boldsymbol{Q}$ =diagonal matrices resulting from placing the vectors $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, $\boldsymbol{l}$ and $\boldsymbol{q}$ in the main diagonal.

x_i, y_i, z_i =coordinates of node i.

$\boldsymbol{x}$,$\boldsymbol{y}$, $\boldsymbol{z}$=vectors of nodal coordinates, x_i, y_i and z_i respectively