22nd International Conference on Automated Planning and Scheduling

June 26, 2012, Atibaia – Sao Paulo – Brazil

SPARK 2012

Proceedings of the Scheduling and Planning Applications woRKshop

Edited by
Luis Castillo Vidal, **Minh Do** and **Riccardo Rasconi**

Organization

Luis Castillo Vidal, IActive, Spain, luis.castillo@iactiveit.com
Minh Do, NASA Ames Research Center / SGT Inc., USA, minh.b.do@nasa.gov
Riccardo Rasconi, ISTC-CNR, Italy, riccardo.rasconi@istc.cnr.it

Program Committee

Susanne Biundo, Universität Ulm, Germany
Mark Boddy, Adventium, USA
Luis Castillo, IActive Intelligent Solutions, Spain
Gabriella Cortellessa, ISTC-CNR, Italy
Mathijs de Weerdt, TU Delft
Minh Do, NASA Ames / SGT Inc., USA
Patrik Haslum, NICTA, Australia
Jana Koehler, IBM Zurich, Switzerland
Robert Morris, NASA Ames, USA
Nicola Policella, ESA-ESOC, Germany
Riccardo Rasconi, ISTC-CNR, Italy
David Smith, NASA Ames, USA
Gérard Verfaillie, ONERA, France
Neil Yorke-Smith, American University of Beirut, Lebanon, and SRI International, USA
Terry Zimmerman, SIFT, USA

Contents

Preface

Application domains that entail planning and scheduling (P&S) problems present a set of compelling challenges to the AI planning and scheduling community, from modeling to technological to institutional issues. New real-world domains and problems are becoming more and more frequently affordable challenges for AI. The international Scheduling and Planning Applications woRKshop (SPARK) was established to foster the practical application of advances made in the AI P&S community. Building on antecedent events, SPARK'12 is the sixth edition of a workshop series designed to provide a stable, long-term forum where researchers and practitioners can discuss the applications of planning and scheduling techniques to real-world problems. The series webpage is at http://decsai.ugr.es/~lcv/SPARK/

In the attempt to cover the whole spectrum of the efforts in P&S Application-oriented Research, this year's SPARK edition will categorize all contributions in three main areas, namely P&S Under Uncertainty, Execution & Validation, Novel Domains for P&S, and Emerging Applications for P&S. We are once more very pleased to continue the tradition of representing more applied aspects of the planning and scheduling community and to perhaps present a pipeline that will enable increased representation of applied papers in the main ICAPS conference.

We thank the Program Committee for their commitment in reviewing. We thank the ICAPS'12 workshop and publication chairs for their support.

Edited by
Luis Castillo Vidal, Minh Do and Riccardo Rasconi

Composition of Flow-Based Applications with HTN Planning*

Shirin Sohrabi†
University of Toronto
Toronto, Ontario, Canada

Octavian Udrea, Anand Ranganathan, Anton V. Riabov
IBM T.J. Watson Research Center
Hawthorne, NY, U.S.A.

Abstract

Goal-driven automated composition of software components is an important problem with applications in Web service composition and stream processing systems. The popular approach to address this problem is to build the composition automatically using Artificial Intelligence planning. However, it is shown that some of these popular planning approaches may neither be feasible nor scalable for many real large-scale flow-based applications. Recent advances have proven that the automated composition problem can take advantage of expert knowledge restricting the ways in which different reusable components can be composed. This knowledge can be represented using an extensible composition template or pattern. In prior work, a flow pattern language called Cascade and its corresponding specialized planner have shown the best performance in these domains. In this paper, we propose to address this problem using Hierarchical Task Network (HTN) planning. To this end, we propose an automated approach of creating an HTN-based problem from the Cascade representation of the flow patterns. The resulting technique not only allows us to use the HTN planning paradigm and its many advantages including added expressivity but also enables optimization and customization of composition with respect to preferences and constraints. Further, we propose and develop a lookahead heuristic and show that it significantly reduces the planning time. We have performed extensive experimentation in the context of the stream processing application and evaluated applicability and performance of our approach.

Introduction

One of the approaches to automated software composition focuses on composition of information flows from reusable software components. This flow-based model of composition is applicable in a number of application areas, including Web service composition and stream processing. There are a number of tools (e.g., Yahoo Pipes and IBM Mashup Center) that support the modeling of the data flow across multiple components. Although these visual tools are fairly popular, the use of these tools becomes increasingly difficult as the number of available components increases, even more so, when there are complex dependencies between components, or other kinds of constraints in the composition.

While automated Artificial Intelligence (AI) planning is a popular approach to automate the composition of components, Riabov and Liu have shown that Planning Domain Definition Language (PDDL)-based planning approach may neither be feasible nor scalable when it comes to addressing real large-scale stream processing systems or other flow-based applications (e.g., (Riabov and Liu 2006)). The primary reason behind this is that while the problem of composing flow-based applications can be expressed in PDDL, in practice the PDDL-based encoding of certain features poses significant limitation to the scalability of planning.

In 2009, we proposed a pattern-based composition approach where composition patterns were specified using our proposed language called Cascade and the plans were computed using our specialized planner, MARIO (Ranganathan, Riabov, and Udrea 2009). We made use of the observation that automated composition problem can take advantage of expert knowledge of how different components can be coupled together and this knowledge can be expressed using a composition pattern. For software engineers, who are usually responsible for encoding composition patterns, doing so in Cascade is easier and more intuitive than in PDDL or in other planning specification languages. The MARIO planner achieves fast composition times due to optimizations specific to Cascade, taking advantage of the structure of flow-based composition problems, while limiting expressivity of domain descriptions.

In this paper, we propose a planning approach based on Hierarchical Task Networks (HTNs) to address the problem of automated composition of components. To this end, we propose a novel technique for creating an HTN-based planning problem with preferences from the Cascade representation of the patterns together with a set of user-specified Cascade goals. The resulting technique enables us to explore the advantages of using domain-independent planning and HTN planning including added expressivity, and address optimization and customization of composition with respect to preferences and constraints. We use the preference-based HTN planner **HTNPLAN-P** (Sohrabi, Baier, and McIlraith 2009) for implementation and evaluation of our approach. Moreover, we develop a new lookahead heuristic by drawing inspirations from ideas proposed in (Marthi, Russell, and Wolfe 2007). We also propose an algorithm to derive indexes required by our proposed heuristic.

*This paper also appears in the AAAI-12 Workshop on Problem Solving using Classical Planners (CP4PS), 2012.

†This work was done at IBM T.J.Watson Research Center.

The contributions of this paper are as follows: (1) we exploit HTN planning with preferences to address modeling, computing, and optimizing the composition of information flows in software components; (2) we develop a method to automatically translate Cascade patterns into HTN domain description and Cascade goals into preferences, and to that end we address several unique challenges that hinder planner performance in flow-based applications; (3) we perform extensive experiments with real-world patterns using IBM InfoSphere Streams applications; and (4) we develop an enhanced lookahead heuristic that improves HTN planning performance by 65% on average in those applications.

Preliminaries

Specifying Patterns in Cascade

The Cascade language has been proposed in (Ranganathan, Riabov, and Udrea 2009) for specifying flow patterns. A Cascade flow pattern describes a set of flows by describing different possible structures of flow graphs, and possible components that can be part of the graph. Components in Cascade can have zero or more input ports and one or more output ports. A component can be either primitive or composite. A primitive component embeds a code fragment from a flow-based language (e.g., SPADE (Gedik et al. 2008)). These code fragments are used to convert a flow into a program/script that can be deployed on a flow-based information processing platform. A composite component internally defines a flow of other components.

Figure 1 shows an example of a flow pattern, defining a composite called *StockBargainIndexComputation*. Source data can be obtained from either *TAQTCP* or *TAQFile*. This data can be filtered by either a set of tickers, by an industry, or neither as the filter components is optional (indicated by the "?"). The VWAP and the Bargain Index calculations can be performed by a variety of concrete components (which inherit from abstract components *CalculateVWAP* and *CalculateBargainIndex* respectively). The final results can be visualized using a table, a time- or a stream-plot. Note, the composite includes a sub-composite *BIComputationCore*.

A single flow pattern defines a number of actual flows. As an example, let us assume there are 5 different descendants for each of the abstract components. Then, the number of possible flows defined by *StockBargainIndexComputation* is $2 \times 3 \times 5 \times 5 \times 3$, or 450 flows.

A flow pattern in Cascade is a tuple $F = (\mathcal{G}(\mathcal{V}, \mathcal{E}), M)$, where $\mathcal{G}$ is a directed acyclic graph, and M is a main composite. Each vertex, $v \in \mathcal{V}$, can be the invocation of one or more of the following: (1) a primitive component, (2) a composite component, (3) a choice of components, (4) an abstract component with descendants, (5) a component, optionally. Each directed edge, $e \in \mathcal{E}$ in the graph represents the transfer of data from an output port of one component to the input port of another component. Throughout the paper, we refer to edges as **streams**, outgoing edges as "output streams", and ingoing edges as "input streams". The main composite, M, defines the set of allowable flows. For example, if *StockBargainIndexComputation* is the main composite in Figure 1, then any of the 450 flows that it defines can potentially be deployed on the underlying platform.

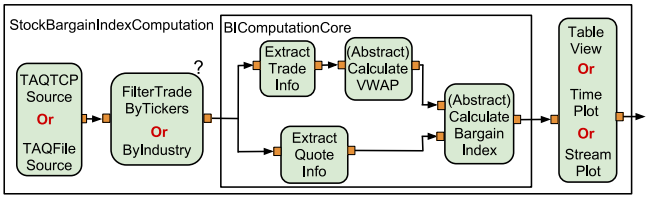


Figure 1: Example of a Cascade flow pattern.

In Cascade, output ports of components (output streams) can be annotated with tags to describe the properties of the produced data. Tags can be any keywords related to terms of the business domain. Tags are used by the end-user to specify the composition goals; we refer to as the **Cascade goals**. For each graph composed according to the pattern, tags associated with output streams are propagated downstream, recursively associating the union of all input tags with outputs for each component. Cascade goals are then matched to the description of graph output. Graphs that include all goal tags become candidate flows (or **satisfying flows**) for the goal. For example, if we annotate the output port of the *FilterTradeByIndustry* component with the tag *ByIndusty*, there would be $2 \times 5 \times 5 \times 3$, or 150 satisfying flows for the Cascade goal "ByIndustry". Planning is used to find "best" satisfying flows efficiently from the millions of possible flows, present in a typical domain.

Hierarchical Task Network (HTN) Planning

HTN planning is a widely used planning paradigm and many domain-independent HTN planners exist (Ghallab, Nau, and Traverso 2004). The HTN planner is given the HTN planning problem: the initial state s_0, the initial task network w_0, and the planning domain D (a set of operators and methods). HTN planning is performed by repeatedly decomposing tasks, by the application of methods, into smaller and smaller subtasks until a primitive decomposition of the initial task network is found. A task network is a pair (U, C) where U is a set of tasks and C is a set of constraints. A task is *primitive* if its name matches with an operator, otherwise it is *nonprimitive*. An operator is a regular planning action. It can be applied to accomplish a primitive task. A method is described by its name, the task it can be applied to $task(m)$, and its task network $subtasks(m)$. A method m can accomplish a task t if there is a substitution σ such that $\sigma(t) = task(m)$. Several methods can accomplish a particular nonprimitive task, leading to different decompositions of it. Refer to (Ghallab et al. 2004) for more information.

HTNPLAN-P (Sohrabi et al. 2009) is a provably optimal preference-based planner, built on top of a Lisp implementation of **SHOP2** (Nau et al. 2003), a highly-optimized HTN planner. **HTNPLAN-P** takes as input an HTN planning problem, specified in the **SHOP2**'s specification language (not in PDDL). **HTNPLAN-P** performs incremental search and uses variety of different heuristics including the Lookahead Heuristic (LA). We modified **HTNPLAN-P** to implement our proposed heuristic, the Enhanced Lookahead Heuristic (ELA). We also use **HTNPLAN-P** to evaluate our approach.

From Cascade Patterns to HTN Planning

In this section, we describe an approach to create an HTN planning problem with preferences from any Cascade flow pattern and goals. In particular, we show how to: (1) create an HTN planning domain from the definition of Cascade components (2) represent the Cascade goals as preferences. We refer to the **SHOP2**'s specification language (also **HTNPLAN-P**'s input language) in Lisp. We consider ordered and unordered task networks specified by keywords ":ordered" and ":unordered", distinguish operators by the symbol "!" before their names, and variables by the symbol "?" before their names.

Creating the HTN Planning Domain

In this section, we describe an approach to translate the different elements and unique features of Cascade flow patterns to operators or methods, in an HTN planning domain.

Creating New Streams One of the features of stream processing domains is that components produce one or more new data streams from several existing ones. Further, the precondition of each input port is only evaluated based on the properties of connected streams; hence, instead of a global state, the state of the world is partitioned into several mutually independent ones. Although it is possible to encode parts of these features in PDDL, the experimental results in (Riabov and Liu 2005; 2006) show poor performance of planners (on an attempt to formulate the problem in PDDL). We believe the main difficulty in the PDDL representation is the ability to address creating new objects that have not been previously initialized to represent the generation of new streams. This can result in a large number of symmetric objects, significantly slowing down the planner.

To address the creation of new uninitialized streams we propose to use the *assignment expression*, available in **SHOP2**'s input language, in the precondition of the operator that creates the new stream (will discuss how to model Cascade components next). We use numbers to represent the stream variables using a special predicate called *sNum*. We then increase this number by manipulating the add and delete effects of the operators that are creating new streams. This *sNum* predicate acts as a *counter* to keep track of the *current* value that we can assign for the new output streams.

The assignment expression takes the form "(assign v t)" where v is a variable, and t is a term. Here is an example of how we implement this approach for the "bargainIndex" stream, the outgoing edge of the abstract component *CalculateBargainIndex* in Figure 1. The following precondition, add and delete list belong to the corresponding operators of any concrete component of this abstract component.

```
Pre:((sNum ?current)(assign ?bargainIndex ?current)
     (assign ?newNum (call  + 1 ?current)))
Delele List: ((sNum ?current))
Add List:    ((sNum ?newNum))
```

Now for any invocation of the abstract component *CalculateBargainIndex*, new numbers, hence, new streams are used to represent the "bargainIndex" stream.

Tagging Model for Components Output ports of components are annotated with tags to describe the properties of the produced data. Some tags are called *sticky* tags, meaning that these properties propagate to all downstream components unless they are *negated* or removed explicitly. The set of tags on each stream depends on all components that appear before them or on all *upstream* output ports.

To represent the association of a tag to a stream, we use a predicate "(Tag Stream)", where *Tag* is a variable or a string representing a tag (must be grounded before any evaluation of state with respect to this predicate), and *Stream* is the variable representing a stream. To address propagation of tags, we use a *forall expression*, ensuring that all tags that appear in the input streams propagate to the output streams unless they are negated by the component. A forall expression in **SHOP2** is of the form "(forall X Y Z)", where X is a list of variables in Y, Y is a logical expression, Z is a list of logical atoms. Here is an example going back to Figure 1. *?tradeQuote* and *?filteredTradeQuote* are the input and output stream variables respectively for the *FilterTradeQuoteByIndustry* component. Note, we know all tags ahead of time and they are represented by the predicate "(tags ?tag)". Also we use a special predicate *diff* to ensure the negated tag "AllCcompanies" does not propagate downstream.

```
(forall (?tag)(and (tags ?tag) (?tag ?QuoteInfo)
                   (diff ?tag AllCompanies))
            ((?tag ?filteredTradeQuote)))
```

Tag Hierarchy Tags used in Cascade belong to tag hierarchy (or tag taxonomies). This notion is useful in inferring additional tags. In the example in Figure 1, we know that the "TableView" tag is a sub-tag of the tag "Visualizable", meaning that any stream annotated with the tag "TableView" is also implicitly annotated by the tag "Visualizable". To address the tag hierarchy we use **SHOP2** axioms. **SHOP2** axioms are generalized versions of Horn clauses, written in this form (:- *head tail*). *Tail* can be anything that appears in the precondition of an operator or a method. The following are axioms that express the hierarchy of views.

```
:- (Visualizable ?stream)((TableView ?stream))
:- (Visualizable ?stream)((StreamPlot ?stream))
```

Component Definition in the Flow Pattern Next, we put together the different pieces described so far in order to create the HTN planning domain. In particular, we represent the abstract components by nonprimitive tasks, enabling the use of methods to represent concrete components. For each concrete component, we create new methods that can decompose this nonprimitive task (i.e., the abstract component). If no method is written for handling a task, this is an indication that the abstract component had no children.

Components can inherit from other components. The net (or expanded) description of an inherited component includes not only the tags that annotate its output ports but also the tags defined by its parent. We represent this inheritance model directly on each method that represents the inherited component using helper operators that add to the output stream, the tags that belong to the parent component.

We encode each primitive component as an HTN operator. The parameters of the HTN operator correspond to the input and output stream variables of the primitive component. The preconditions of the operator include the "assign expressions" as mentioned earlier to create new output

streams. The add list also includes the tags of the output streams if any. The following is an HTN operator that corresponds to the *TableView* primitive component.

```
Operator: (!TableView ?bargainIndex  ?output)
Pre: ((sNum ?current) (assign ?output ?current)
      (assign ?newNum (call + 1 ?current)))
Delete List: ((sNum ?current))
Add List:((sNum ?newNum)(TableView ?bargainIndex)
          (forall (?tag) (and (tags ?tag)
           (?tag ?bargainIndex))((?tag ?output))
```

We encode each composite component as HTN methods with task networks that are either ordered or unordered. Each composite component specifies a *graph clause* within its body. The corresponding method addresses the graph clause using task networks that comply with the ordering of the components. For example, the graph clause within the *BIComputationCore* composite component in Figure 1 can be encoded as the following task. Note the parameters are omitted. Note also, we used ordered task networks for representing the sequence of components, and an unordered task network for representing the split in the data flow.

```
(:ordered (:unordered (!ExtractQuoteInfo)
   (:ordered (!ExtractTradeInfo) (CalculateVWAP)))
  (CalculateBargainIndex))
```

Structural Variations of Flows There are three types of structural variation in Cascade: enumeration, optional components, and use of high-level components. Structural variations create patterns that capture multiple flows. Enumerations are specified by listing the different possible components. To capture this we use multiple methods applicable to the same task. A component can be specified as optional, meaning that it may not appear as part of the flow. We capture optional components using methods that simulate the *no-op* task. Abstract components are used in flow patterns to capture high-level components. These components can be replaced by their concrete components. In HTN, this is already captured by the use of nonprimitive tasks for abstract components and methods for each concrete component.

Specifying Cascade Goals as Preferences

While Cascade flow patterns specify a set of flows, users can be interested in only a subset of these. Thus, users are able to specify the Cascade goals by providing a set of tags that they would like to appear in the final stream. We propose to specify the user-specified Cascade goals as Planning Domain Definition Language (PDDL3) (Gerevini et al. 2009) simple preferences. **Simple preferences** are atemporal formulae that express a preference for certain conditions to hold in the final state of the plan. In PDDL3 the quality of the plan is defined using a metric function. The PDDL3 function `is-violated` is used to assign appropriate weights to different preference formula. Note, inconsistent preferences are automatically handled by the metric function.

The advantage of encoding the Cascade goals as preferences is that the users can specify them outside the domain description as an additional input to the problem. Also, by encoding the Cascade goals as preferences, if the goals are not achievable, a solution can still be found but with an associated quality measure. In addition, the preference-based planner, **HTNPLAN-P**, can potentially guide the planner towards achieving these preferences; can do branch and bound with sound pruning using admissible heuristics, whenever possible to guide the search toward a high-quality plan.

The following are some example. If the Cascade goals encoded as preferences are mutually inconsistent, we can assign a higher weight to the "preferred" goal. Otherwise, we can use uniform weights when defining a metric function.

```
(preference g1 (at end (ByIndustry ?finalStream)))
(preference g2 (at end (TableView ?finalStream)))
(preference g3 (at end (LinearIndex ?finalStream)))
```

Flow-Based HTN Planning Problem with Preferences

In this section, we characterize a flow-based HTN planning problem with preferences and discuss the relationship between satisfying flows and optimal plans.

A Cascade flow pattern problem is a 2-tuple $P^F = (F, G)$, where $F = (\mathcal{G}(\mathcal{V}, \mathcal{E}), M)$ is a Cascade flow pattern (where $\mathcal{G}$ is a directed acyclic graph, and M is the main composite), and G is the set of Cascade goals. α is a satisfying flow for P^F if and only if α is a flow that meets the main composite M. Set of Cascade goals G is realizable if and only if there exists at least one satisfying flow for it.

Given the Cascade flow pattern problem P^F, we define the corresponding flow-based HTN planning problem with preferences as a 4-tuple $P = (s_0, w_0, D, \preceq)$, where: s_0 is the initial state consisting of a list of all tags and our special predicates; w_0 is the initial task network encoding of the main component M; D is the HTN planning domain, consisting of a set of operators and methods derived from the Cascade components $v \in \mathcal{V}$; and $\preceq$ is a preorder between plans dictated by the set of Cascade goals G.

Proposition 1 *Let $P^F = (F, G)$ be a Cascade flow pattern problem where G is realizable. Let $P = (s_0, w_0, D, \preceq)$ be the corresponding flow-based HTN planning problem with preferences. If α is an optimal plan for P, then we can construct a flow (based on α) that is a satisfying flow for the problem P^F.*

Consider the Cascade flow pattern problem P^F with F shown in Figure 1 and G be the "TableView" tag. Let P be the corresponding flow-based HTN problem with preferences. Then consider the following optimal plan for P: [TAQFileSource(1), ExtradeTradeInfo(1,2), VWAPByTime(2,3), ExtractQuoteInfo(1,4), BISimple(3,4,5), TableView(5,6)]. We can construct a flow in which the components mentioned in the plan are the vertices and the edges are determined by the numbered parameters corresponding to the generated output streams. The resulting graph is not only a flow but a satisfying flow for the problem P^F.

Computation

In the previous section, we described a method that translates Cascade flow patterns and Cascade goals into an HTN planning problem with preferences. We also showed the relationship between optimal plans and satisfying flows. Now given a specification of preference-based HTN planning in hand we select **HTNPLAN-P** to compute these optimal plans that later get translated to satisfying flows for the original Cascade flow patterns. In this section, we focus on our pro-

posed heuristic, and describe how the required indexes for this heuristic can be generated in the preprocessing step.

Enhanced Lookahead Heuristic (*ELA*)

The enhanced lookahead function estimates the metric value achievable from a search node N. To estimate this metric value, we compute a set of reachable tags for each task within the initial task network. A set of tags are reachable by a task if they are reachable by any **plan** that extends from decomposing this task. Note, we assume that every nonprimitive task can eventually have a primitive decomposition.

The *ELA* function is an underestimate of the actual metric value because we ignore deleted tags, preconditions that may prevent achieving a certain tag, and we compute the set of all reachable tags, which in many cases is an overestimate. Nevertheless, this does not necessarily mean that *ELA* function is a lower bound on the metric value of any plan extending node N. However, if it is a lower bound, then it will provide sound pruning (following Baier et al. 2009) if used within the **HTNPLAN-P** search algorithm and provably optimal plans can get generated. A pruning strategy is sound if no state is incorrectly pruned from the search space. That is whenever a node is pruned from the search space, we can prove that the metric value of any plan extending this node will exceed the current bound best metric. To ensure that the *ELA* is monotone, for each node we take the intersection of the reachable tags computed for this node's task and the set of reachable tags for its immediate predecessor.

Proposition 2 *The ELA function provides sound pruning if the preferences are all PDDL3 simple preferences and the metric function is non-decreasing in the number of violated preferences and in plan length.*

Our notion of reachable tags is similar to the notion of "complete reachability set" in Marthi et al. (2007). While they find a superset of all reachable states by a "high-level" action a, we find a superset of all reachable tags by a task t; this can be helpful in proving a certain task cannot reach a goal. However, they assume that for each task a sound and complete description of it is given in advance, whereas we do not assume that. In addition, we are using this notion of reachability to compute a heuristic, which we implement in **HTNPLAN-P**. They use this notion for pruning plans and not necessarily in guiding the search towards a preferred plan.

Generation from HTN

In this section, we briefly discuss how to generate the reachable tags from the corresponding HTN planning problem. Algorithm 1 shows pseudocode of our offline procedure that creates a set of reachable tags for each task. It takes as input the planning domain D, a set of tasks (or a single task) w, and a set of tags to carry over C. The algorithm is called initially with the initial task network w_0, and $C = \emptyset$. To track the produced tags for each task we use a map R. If w is a task network then we consider three cases: 1) task network is empty, we then return C, 2) w is an ordered task network, then for each task t_i we call the algorithm starting with the right most task t_n updating the carry C, 3) w is unordered, then we call GetRTags twice, first to find out what each task produces (line 8), and then again with the updated set of carry tags (line 10). This ensures that we overestimate the reachable tags regardless of the execution order.

Algorithm 1: The GetRTags (D, w, C) algorithm.

```
1  initialize global Map R;  T ← ∅;
2  if w is a task network then
3      if w = ∅ then return C;
4      else if w = (:ordered t_1 ... t_n) then
5          for i=n to 1 do  C ← GetRTags(D, t_i, C);
6      else if w = (:unordered t_1 ... t_n) then
7          for i=1 to n do
8              T_{t_i} ← GetRTags(D, t_i, ∅); T ← T_{t_i} ∪ T;
9          for i=1 to n do
10             C_{t_i} ← ⋃_{j=1, j≠i}^{n} T_j ∪ C; GetRTags(D, t_i, C_{t_i});
11 else if w is a task then
12     if R[w] is not defined then R[w] ← ∅;
13     else if t is primitive then T ← add-list of an operator that matches;
14     else if t is nonprimitive then
15         M' ← {m_1, ..., m_k} such that task(m_i) match with t;
16         U' ← {U_1, ..., U_k} such that U_i = subtask(m_i);
17         foreach U_i ∈ U' do T ← GetRTags(D, U_i, C) ∪ T;
18     R[w] ← R[w] ∪ T ∪ C;
19 return T ∪ C
```

If w is a task then we update its returned value $R[w]$. If w is primitive, we find a set of tags it produces by looking at its add-list. If w is nonprimitive then we first find all the methods that can be applied to decompose it and their associated task networks. We then take a union of all tags produced by a call to GetRTags for each of these task networks.

Our algorithm can be updated to deal with recursive tasks by first identifying when loops occur and then by modifying the algorithm to return special tags in place of a recursive task's returned value. We then use a fixed-point algorithm to remove these special tags and update the values for all tasks.

Experimental Evaluation

We had two main objectives in our experimental analysis: (1) evaluate the applicability of our approach when dealing with large real-world applications or composition patterns, (2) evaluate the computational time gain that may result from use of our proposed heuristic. To address our first objective, we took a suite of diverse Cascade flow pattern problems from patterns described by customers for IBM InfoSphere Streams and applied our techniques to create the corresponding HTN planning problems with preferences. We then examined the performance of **HTNPLAN-P**, on the created problems. To address our second objective, we implemented the preprocessing algorithm discussed earlier and modified **HTNPLAN-P** to incorporate the enhanced lookahead heuristic within its search strategy and then examined its performance. A search strategy is a prioritized sequence of heuristics that determines if a node is better than another.

We had 7 domains and more than 50 HTN planning problems in our experiments. The created HTN problems come from patterns of varying sizes and therefore vary in hardness. For example, a problem can be harder if the pattern had many optional components or many choices, hence influencing the branching factor. Also a problem can be harder if the tags that are part of the Cascade goal appear in the harder to reach branches depending on the planner's search strategy. For **HTNPLAN-P**, it is harder if the goal tags appear

Dom	Prob	Plan Length	# of Plans	*No-LA* Time (s)	*LA* Time (s)
1	1	11	81	0.04	0.05
	2	11	162	0.10	**0.01**
	3	11	81	0.18	**0.04**
2	1	11	162	0.04	0.05
	2	11	162	0.13	**0.01**
	3	11	81	0.25	**0.04**
3	1	38	2^{26}	0.08	0.08
	2	38	2^{13}	276.11	**0.09**
	3	20	2^{13}	OM	**0.14**
	4	38	2^{26}	OM	**0.14**
4	1	44	4608^{2}	0.09	0.11
	2	92	4608^{4}	0.64	**0.61**
	3	184	4608^{8}	4.80	**4.50**
	4	368	4608^{16}	43.00	**35.00**

Figure 2: Evaluating the applicability of our approach by running **HTNPLAN-P** (two modes) as we increase problem hardness.

in the very right side of the search space since it explores the search space from left to right if the heuristic is not informing enough. All problems were run for 10 minutes, and with a limit of 1GB per process. "OM" stands for "out of memory", and "OT" stands for "out of time".

We show a subset of our results in Figure 2. Columns 5 and 6 show the time in seconds to find an optimal plan. We ran **HTNPLAN-P** in its existing two modes: *LA* and *No-LA*. *LA* means that the search makes use of the *LA* (lookahead) heuristic (*No-LA* means it does not). Note **HTNPLAN-P**'s other heuristics are used to break ties in both modes.We measure plan length for each solved problem as a way to show the number of generated output streams. We show the number of possible optimal plans for each problem as an indication of the size of the search space. This number is a lower bound in many cases on the actual size of the search space. Note we only find one optimal plan for each problem through the incremental search performed by **HTNPLAN-P**.

The results in Figure 2 indicates the applicability and feasibility of our approach as we increase the difficulty of the problem. All problems were solved within 35 seconds by at least one of the two modes used. The result also indicates that not surprisingly, the *LA* heuristic performs better at least in the harder cases (indicated in bold). This is partly because the *LA* heuristic forms a sampling of the search space. In some cases, due to the possible overhead in calculation of the *LA* heuristic, we did not see an improvement. Note that in some problems (3rd domain Problems 3 and 4), an optimal plan was only found when the *LA* heuristic was used.

We had two sub-objectives in evaluating our proposed heuristic, the Enhanced Lookahead Heuristic (*ELA*): (1) to find out if it improves the time to find an optimal plan (2) to see if it can be combined with the planner's previous heuristics, namely the *LA* heuristic. To address our objectives, we identified cases where **HTNPLAN-P** has difficulty finding the optimal solution. In particular we chose the third and fourth domain and tested with goal tags that appear deep in the right branch of the HTN search tree. These problems are difficult because achieving the goal tags are harder and the *LA* heuristic fails in providing sufficient guidance.

Figure 3 shows a subset of our results. *LA* then *ELA* (resp. *ELA* then *LA*) column indicates that we use a strategy in which we compare two nodes first based on their *LA* (resp. *ELA*) values, then break ties using their *ELA* (resp. *ELA*) values. In the Just *ELA* and Just *LA* columns we used either just *LA* or *ELA*. Finally in the *No-LA* column we did not use either heuristics. Our results show that the ordering of the heuristics does not seem to make any significant change in the time it takes to find an optimal plan. The results also show that using the *ELA* heuristic improves the search time compared to other search strategies. In particular, there are cases in which the planner fails to find the optimal plan when using *LA* or *No-LA* but the optimal plan is found within the tenth of a second when using the *ELA* heuristic. To measure the gain in computation time from the *ELA* heuristic technique, we computed the percentage difference between the *LA* heuristic and the *ELA* heuristic times, relative to the worst time. We assigned a time of 600 to those that exceeded the time or memory limit. The results show that on average we gained 65% improvement when using *ELA* for the problems we used. This shows that our enhanced lookahead heuristic seems to significantly improve the performance.

Dom	Prob	*LA* then *ELA* Time (s)	*ELA* then *LA* Time (s)	Just *ELA* Time (s)	Just *LA* Time (s)	*No-LA* Time (s)
3	5	1.70	1.70	**0.07**	0.13	OM
	6	1.70	1.70	**0.07**	1.50	OM
	7	1.80	1.80	**0.07**	1.60	OM
	8	1.70	1.70	**0.07**	OM	OM
	9	1.40	1.40	**0.07**	OM	OM
	10	1.40	1.30	**0.07**	OM	OM
4	5	0.58	0.45	**0.02**	0.56	0.12
	6	2.28	2.24	**0.07**	3.01	0.38
	7	14.40	14.28	**0.44**	19.71	1.44
	8	104.70	102.83	**3.15**	147.00	8.00
	9	349.80	341.20	**10.61**	486.53	18.95
	10	OT	OT	**24.45**	OT	40.20

Figure 3: Evaluation of the *ELA* heuristic.

Summary and Related Work

There is a large body of work that explores the use of AI planning for the task of automated Web service composition (e.g., (Pistore et al. 2005)). Additionally some explore the use of some form of expert knowledge (e.g., (McIlraith and Son 2002)). While similarly, many explore the use of HTN planning, they rely on the translation of OWL-S (Martin et al. 2007) service descriptions of services to HTN planning (e.g., (Sirin et al. 2005)). Hence, the HTN planning problems driven from OWL-S generally ignore the data flow aspect of services, a major focus of Cascade flow patterns.

In this paper, we examined the correspondence between HTN planning and automated composition of flow-based applications. We proposed use of HTN planning and to that end proposed a technique for creating an HTN planning problem with user preferences from Cascade flow patterns and user-specified Cascade goals. This opens the door to increased expressive power in flow pattern languages such as Cascade, for instance the use of recursive structures (e.g., loops), user preferences, and additional composition constraints. We also developed a lookahead heuristic and showed that it improves the performance of **HTNPLAN-P** for the domains we used. The proposed heuristic is general enough to be used within other HTN planners. We have performed extensive experimentation that showed applicability and promise of the proposed approach.

References

Baier, J. A.; Bacchus, F.; and McIlraith, S. A. 2009. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence* 173(5-6):593–618.

Gedik, B.; Andrade, H.; lung Wu, K.; Yu, P. S.; and Doo, M. 2008. SPADE: the System S declarative stream processing engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1123–1134.

Gerevini, A.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence* 173(5–6):619–668.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Hierarchical Task Network Planning. Automated Planning: Theory and Practice*. Morgan Kaufmann.

IBM. IBM InfoSphere Streams. http://www.ibm.com/software/data/infosphere/streams/. [online; accessed 14-05-2012].

IBM. IBM Mashup Center. http://www-01.ibm.com/software/info/mashup-center/. [online; accessed 14-05-2012].

Marthi, B.; Russell, S. J.; and Wolfe, J. 2007. Angelic semantics for high-level actions. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 232–239.

Martin, D.; Burstein, M.; McDermott, D.; McIlraith, S.; Paolucci, M.; Sycara, K.; McGuinness, D.; Sirin, E.; and Srinivasan, N. 2007. Bringing semantics to Web services with OWL-S. *World Wide Web Journal* 10(3):243–277.

McIlraith, S., and Son, T. 2002. Adapting Golog for composition of semantic Web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR)*, 482–493.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Pistore, M.; Marconi, A.; Bertoli, P.; and Traverso, P. 2005. Automated composition of Web services by planning at the knowledge level. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 1252–1259.

Ranganathan, A.; Riabov, A.; and Udrea, O. 2009. Mashup-based information retrieval for domain experts. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM)*, 711–720.

Riabov, A., and Liu, Z. 2005. Planning for stream processing systems. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 1205–1210.

Riabov, A., and Liu, Z. 2006. Scalable planning for distributed stream processing systems. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 31–41.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2005. HTN planning for Web service composition using SHOP2. *Journal of Web Semantics* 1(4):377–396.

Sohrabi, S.; Baier, J. A.; and McIlraith, S. A. 2009. HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1790–1797.

Yahoo. Yahoo pipes. http://pipes.yahoo.com. [online; accessed 14-05-2012].

Planning and Scheduling Ship Operations on Petroleum Ports and Platforms

Tiago Stegun Vaquero[1] and **Gustavo Costa**[2] and **Flavio Tonidandel**[3]
Haroldo Igreja[4] and **José Reinaldo Silva**[2] and **J. Christopher Beck**[1]
[1]Department of Mechanical & Industrial Engineering, University of Toronto, Canada
[2]Department of Mechatronics Engineering, University of São Paulo, Brazil
[3]IAAA Lab, University Center of FEI - São Bernardo do Campo, Brazil
[4]Transpetro, PETROBRAS, Brazil
{tvaquero,jcb}@mie.utoronto.ca, {gustavorochacosta, reinaldo}@usp.br, flaviot@fei.edu.br, higreja@petrobras.com.br

Abstract

In this paper, we address the process of modeling planning and scheduling ship operations on petroleum platforms and ports. The general problem to be solved is based on the transportation and delivery of a list of requested cargo to different locations considering a number of constraints and elements based on a real problem of Petrobras – the Brazilian Petroleum Company. The objective is to optimize a set of costs brought by the execution of a schedule. Modeling the problem in UML and then translating to PDDL is shown to be feasible and practical by using itSIMPLE. However, although domain-independent planners can provide valid solutions to simplified versions of the problem, they struggle with a more realistic version.

Introduction

With the discovery of a promising massive oilfield beneath 2000 to 3000 meters of water in 2007, the Brazilian government has been investing in advanced technologies and infrastructure for deep water extraction of oil and natural gas. New discoveries in what is called the *pre-salt* basin created even more challenges in deep water exploitation and in several underlying engineering problems in order to make this effort secure, profitable and safe for the environment. One of the challenges is the planning and scheduling of vessels which transport goods, components and tools between crowded ports on land to platforms in the ocean. The supply of these elements to the network of platforms is essential to maintaining a fully operational oil extraction station off the Brazilian coast. Potential expansion of the number of platforms must be carefully studied and optimized to result in minimal impact on the environment. Hence, studying the planning and scheduling of ship operations in those ports and platforms is one of the aims of Petrobras.

The general problem to be solved is based on the transportation and delivery of a list of requested cargo to different locations considering a number of constraints and elements such as available ports, platforms, vessel capacity, weights of cargo items, fuel consumption, available refueling stations in the ocean, different duration of operations, and costs. Given a set of cargo items, the problem is to find a feasible plan that guarantees their delivery while respecting the constraints and requirements of the ship capacities. The objective is to minimize the total amount of fuel used, the size of waiting queues in ports, the number of ships used, the makespan of the schedule and the docking cost. The problem has a number of features that have been addressed by heuristic-based space-state search. Thus, it is a realistic problem that may be amenable to planning technology.

Since the 1980s there has been a recurring discussion in the literature regarding the relationship between Artificial Intelligence (AI) planning and optimization problems. A particular contrast is the traditional satisfaction-oriented bias of AI planning (Kautz and Walser 1999; 2000) versus the substantial focus and exploitation of cost functions in optimization approaches studied in Operations Research. Developing solvers for planning & scheduling (P&S) applications that demand both satisfaction-oriented approaches and optimization mechanisms is, with the current technology, still challenging. This is also a challenge faced by Knowledge Engineering (KE) tools and approaches: how to allow designers (both problem-domain experts and planning experts) to model problems requiring sophisticated planning capabilities, reasoning about time constraints, and the expression and minimization of complicated cost functions. There are not many KE tools available for modeling these sorts of problems in the AI P&S literature (Vaquero, Silva, and Beck 2011). As a consequence, the problem presented in this paper is one of the challenge domains in the Fourth International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012).

In this paper, we describe the modeling process we undertook using an AI P&S approach to study one potential expansion of the network of platforms. Our aim is (1) to investigate and describe the modeling process of the ship operation problem in such a network utilizing KE tools (in this case, the itSIMPLE tool (Vaquero et al. 2009)) and standard languages from AI P&S (e.g., PDDL), and (2) to study the use of available domain-independent planners and their performance in solving the model and generating plans. Even though we do not use real data in this paper due to privacy policies, it does not change or reduce the challenge of modeling and solving the problem. The main contributions of this work are: the design of a knowledge model for the planning and scheduling problem of ship operations in petroleum ports and platforms following the AI P&S ap-

proach; and experimental studies that explore the performance of domain-independent, heuristic-based planners on a realistic P&S problem that includes numeric variables and time constraints.

This paper is structured as follows. Firstly, we describe the problem, its restrictions and requirements. Secondly, we describe the design process, focusing on the modeling approach using itSIMPLE. Next, we provide experimental results obtained by selected domain-independent planners when solving problem instances of increasing size in two different scenarios: with and without time constraints. We conclude with a discussion of the results.

Problem Description

The problem of planning and scheduling ship operations on petroleum platforms and ports includes vessel capacity restrictions, the optimization of multiple, coupled objectives, and many others features that make this domain a challenge to AI planning systems. The model of this problem was simplified to focus on the need to provide transportation of goods from ports on the land to platforms in the ocean. In this problem, we consider two strips of the Brazilian coast: *Rio de Janeiro* and *Santos*. Each strip has one port (port *P1* at Rio de Janeiro and port *P2* at Santos) where the loading activities of cargo items occur to support petroleum extraction in deep water.

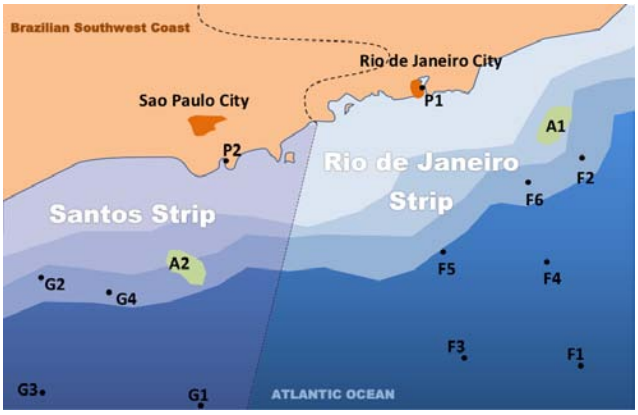


Figure 1: Layout of the strips and position of the ports on the Brazilian Coast.

Both strips contain a set of ocean platforms: six platforms (*F1, ..., F6*) in the Rio de Janeiro strip and four (*G1, ..., G4*) in the Santos strip. The ports are located 200 km from each other while platforms are located from 100 km to 300 km from ports. These platforms frequently require cargo that must be delivered from a port to the requesting platform. Each group of platforms is located in the strip connected to their respective port onshore, as shown in Figure 1. A vessel loads cargo at a port (and sometimes at platforms) and travels to target points for delivery of part or all of its cargo. After completing a delivery, ships go to the waiting areas off-shore. There is one waiting area in each strip: the one in Rio de Janeiro (called *A1*) is located 120 km (radial distance) from port *P1* and the one in Santos (called *A2*) is located 100 km from port *P2*. The distance between *A1* and *A2* is 340 km. Tables 1, 2 and 3 provide the distances between ports, platforms and waiting areas in this problem, as illustrated in Figure 1.

	P1	F2	F3	F4	F5	F6
F1	300km	168km	168km	120km	260km	240km
F2	160km	-	240km	120km	168km	120km
F3	280km	240km	-	120km	168km	260km
F4	200km	120km	120km	-	120km	168km
F5	160km	168km	168km	120km	-	120km
F6	130km	120km	260km	168km	120km	-

Table 1: Distance between platforms and ports in the Rio de Janeiro strip.

	P2	G2	G3	G4
G1	300km	200km	120km	260km
G2	180km	-	260km	120km
G3	280km	260km	-	200km
G4	140km	120km	200km	-

Table 2: Distance between platforms and ports in the Santos strip.

Vessels are the main resource used to transport cargo items from/to ports and platforms. A set of ships is responsible for supplying the platforms. In this problem, we consider ten available vessels (*S1, ..., S10*): six of them have the Rio de Janeiro strip as their base and four of them have Santos as base. Cargo items (*C1, ...,CN*) refer to products, food, equipment, and parts that must be delivered to platforms and/or ports. They are represented as containers in this work.

Given a set of cargo items to transport and their respective locations, the challenge is to find a feasible plan that delivers all cargo properly, minimizing the total amount of fuel used, the makespan and the costs involved. Such a feasible plan must respect the requirements described in the remainder of this section.

Ports and Platforms: The ports can dock two ships simultaneously for loading, unloading and refueling. After receiving two ships, all further requests for docking have to be queued. The cost for docking is is 1000 Brazilian Reais per hour. This cost is applied only when the vessel is moored in a port, and is computed from the time the vessel starts docking to the time it undocks. We do not address the packing and organization of the cargo in the vessel, only the loading/unloading rate.

Besides the port, a vessel can refuel at a subset of the platforms. For this problem, we consider platforms *F5* and *G3* as capable for providing refueling operations. The refueling operation of a vessel is performed at a rate of 100 liters per hour in both ports and platforms. Only one vessel can dock to a platform at any given time.

Vessels: Each ship has a limited capacity for cargo items (100 tons) and a limited fuel tank (600 liters). Traveling with the specified speed average of 70 km/h, ships consume

	F1	F2	F3	F4	F5	F6	A1	A2	P1	P2
G1	468km	580km	420km	500km	380km	520km	540km	320km	350km	300km
G2	580km	468km	380km	520km	300km	500km	540km	110km	400km	180km
G3	588km	600km	420km	560km	580km	580km	580km	400km	450km	280km
G4	600km	588km	580km	580km	420km	580km	570km	180km	420km	140km
A1	200km	40km	320km	280km	180km	80km	-	340km	120km	270km
A2	340km	380km	370km	340km	280km	300km	340km	-	270km	100km
P1	300km	160km	280km	200km	160km	130km	120km	270km	-	200km
P2	380km	290km	320km	340km	270km	300km	270km	100km	200km	-

Table 3: Distance between the platforms, ports and waiting areas in the Rio de Janeiro and Santos strips.

1 liter of fuel each 3 km when traveling fully loaded and 1 liter each 5 km if empty. We assume that all ships have the same capacity for cargo and the same average speed.

Before executing any activity in a port or a platform, ships must perform a docking process. The docking or undocking process of a vessel at a port takes 1 hour, whereas at a platform it takes 0.5 hour. Ships can be docked at ports and platforms to load and unload cargo items, to be refueled, or both. The loading and unloading processes can be done either at the platforms in the ocean or at the port onshore; however, they cannot be done at the same time in a given location. Each vessel can perform the loading/unloading operation with a rate of 1 ton per hour. Refueling can be done at the port or at platforms that have a refueling system, and can be performed during loading or unloading. The rates for refueling are the following: 100 liters per hour at a platform; 100 liters in half an hour at the ports.

Cargo Items: Cargo items can be carried by ships from one location to another, and we disregard the order of loading and unloading in this problem. Since each cargo item has a specified weight, loading a ship is limited by the capacity of that ship. The weight for each cargo item is specified in the request and is considered input data for the problem.

Waiting areas in the ocean: All vessels have to be in a waiting area at the beginning its multiple deliveries. At the end of all deliveries the vessels must go back to a waiting area to wait for the next requests. It is possible to send a vessel located initially in one waiting area to another waiting area of the other strip. However, it is important to have a balanced number of vessels in each one. The ideal balance is 6 vessels in the Rio de Janeiro area *A1* and 4 in the Santos waiting area *A2*.

The use of waiting areas is important to avoid long and unnecessary docking periods at the ports (since there is a cost associated with each docking period) and at the platforms. When parking at the waiting areas, ships must have sufficient fuel to return to a refueling location.

The Modeling Process with itSIMPLE

The KE tool called itSIMPLE (Vaquero et al. 2007; 2009) was used to support the construction and development of the domain model for the problem described above. itSIMPLE's integrated environment focuses on the crucial initial phases of a design.

The tool allows users to follow a disciplined design process to create knowledge intensive models of planning domains, from the informality or semi-informality of real world requirements to formal specifications and domain models that can be read by domain-independent planners (those that read PDDL). The suggested design process for building planning domain models includes the following phases: requirements specification; modeling; model analysis; testing with planners; and plan evaluation (Vaquero et al. 2007). These phases are inherited from Software Engineering and Design Engineering, combined with real planning domain modeling experiences. In this paper, we focus on three of the main phases of such a design process: modeling, testing with planners, and plan analysis.

Domain Modeling

Modeling in itSIMPLE follows an object-oriented approach. Requirements are gathered and modeled using *Unified Modeling Language* (UML) (OMG 2005), a general purpose language broadly accepted in Software Engineering and Requirements Engineering. UML is used to specify, visualize, modify, construct and document domains or artifacts, generally following an object-oriented approach. The tool allows the modeling of a planning problem using diagrams such as *class diagram*, *state machine diagram*, *timing diagram*, and *object diagram*

The class diagram represents the static structure of the planning domain. It shows the existing types of objects, their relationships, properties, operators (actions) and constraints. Class attributes and associations give a visual notion of the semantics of the model. Figure 2 shows the class diagram designed for the Petrobras problem. The diagram consists of nine classes: *Basin*, *Location*, *WaitingArea* (a specialization of Location), *DockingLocation* (also a specialization of Location), *Port* (a specialization of *DockingLocation*), *Platform* (also a specialization of *DockingLocation*), *Cargo*, *Ship*, and *Global* (the class *Global* is a utility class that stores global variables that are accessed from all other classes). The classes illustrated in Figure 2 model all the entities relevant to the problem.

The class *Ship* has several properties that match the requirements. We tried to use straightforward names for these properties to facilitate the understanding of the model and provide an intuitive semantics for a non-planning expert (e.g., *loadcapacity* and *currentload* are numeric values representing the capacity of the ship and its current load); how-

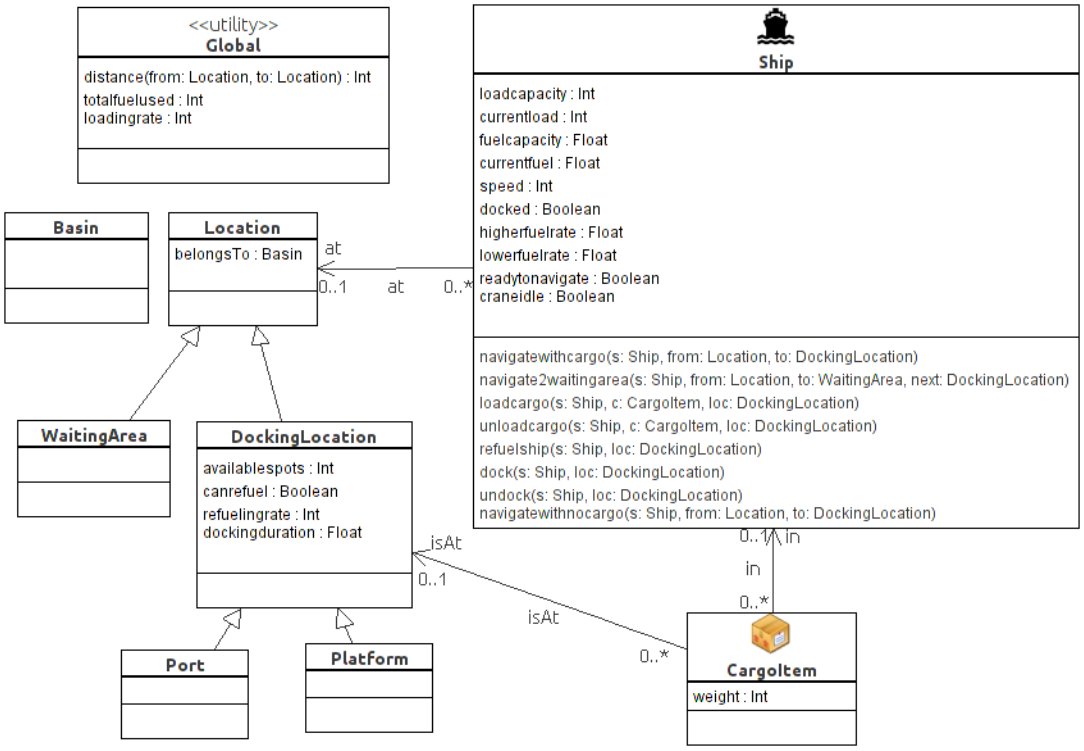


Figure 2: Class diagram of the ship operations problem in petroleum ports and platforms.

ever, some of them deserve further explanation. The variables *higherfuelrate* and *lowerfuelrate* are the fuel consumption rates of the ship when navigating with and without cargo items, respectively. Even though fuel consumption rates are the same for every ship in this problem, we decided to store this information in each ship for extensibility: possible changes in ship performance in a more dynamic environment would require re-planning. The mutually exclusive variables *readytonavigate* and *docked* refer to the status of the ship, whether it is available for moving from one location to another or docked in any docking location (port or platform). Finally, *craneidle* signals if the ship is performing neither loading nor loading operations. It is used to avoid executing them concurrently.

Both *WaitingArea* and *DockingLocation* represent the information about which basin they belong to. Property *availablespots* in the *DockingLocation* is a numeric variable corresponding to how many ships are currently allowed to dock. If a vessel docks at a port or platform, this variable is decreased by 1; it is increased if an undock operation is performed. If an instance of *DockingLocation* can perform a refueling operation, then variable *canrefuel* is set to *true* and a refueling rate can be specified (e.g., 100 liters/hour at a platform and 200 liters/hour at a port). The different (un)docking durations at ports and platforms are specified in the *dockingduration* variable (since docking and undocking durations are the same in a given location, we use *dockingduration* to represent both). Here we also store the (un)docking duration for either location.

The *Global* class holds the information about the distance between the location points shown in Figure 1 and specified in Tables 1, 2 and 3. The total fuel used by ships while delivering the cargo items is stored in the global property *totalfuelused*, which defines the quality of the plan and which must be minimized by the planners. Even though the problem has a set of criteria to be optimized, in this model we evaluate only the total fuel used and the makespan. In addition, *loadingrate* holds the rate of loading and unloading cargo in the ports and platforms (1 ton/h).

We have identified eight main operators (action schema) performed by the ships, as listed below.

- *navigatewithnocargo*: Navigate from one location to a docking location without cargo. Lower fuel consumption is considered. The duration for this action is specified as 'distance(from,to)/s.speed'.
- *navigatewithcargo*: Navigate from one location to a docking location with cargo (*currentload* > 0). Higher fuel consumption is considered. The duration is specified as 'distance(from,to)/s.speed'.
- *navigate2waitingarea*: Navigate from one location to a waiting area. This operator considers the total fuel consumption necessary to get to the destination and then to a refueling location. *lowerfuelrate* is employed in this case. The duration is specified as 'distance(from,to)/s.speed'.
- *dock*: Dock the ship in one of the available spots in the docking location (port or platform). The duration is specified as 'loc.dockingduration'.

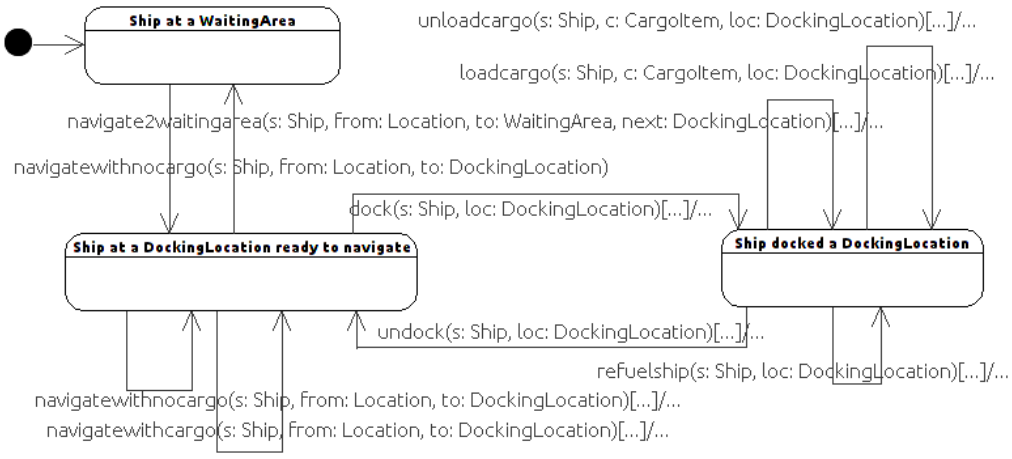


Figure 3: State machine diagram of the Ship.

- *undock*: Undock the ship from one of the spots used in the docking location (port or platform). The crane must be idle for this operation. The duration is specified as 'loc.dockingduration'.
- *loadcargo*: Load a cargo item from the location where the ship is docked. The ship must have available capacity to load the item. The crane must be idle and during the whole operation the crane becomes unavailable. The duration is specified as 'c.weight/loadingrate'.
- *unloadcargo*: Unload a cargo item from the docked ship to the location. The crane must be idle and during the whole operation the crane is unavailable. The duration is specified as 'c.weight/loadingrate'.
- *refuelship*: Refuel the ship's tank to its maximum capacity. The ship must be docked during the whole operation. The duration is specified as '(s.fuelcapacity - s.currentfuel)/loc.refuelingrate' which is the time necessary to re-fill the fuel that has been consumed.

The actions of the domain are modeled using two diagrams: the class diagram and the state machine diagram. In the class diagram, we define the name, parameters and duration for each operator (we use discrete time). The dynamics of the actions are specified in the state machine diagram, in which it is possible to represent the pre- and post-conditions of the operators declared in the class diagram. In itSIMPLE, pre- and post-conditions are defined using the formal constraint language called *Object Constraint Language* (OCL) (OMG 2003), a predefined language of UML. Usually every class in the class diagram has its own state machine diagram. A state machine diagram does not intend to specify all changes caused by an action. Instead, the diagram details only the changes that the action causes in an object of a specific class. Figures 3 and 4 show the state machine diagrams for the classes *Ship* and *Cargo*, respectively.

Timing diagrams and annotated OCL expressions are used to specify how properties change in an action horizon. For example, properties such as *readytonavigate* and *craneidle* become *false* when the action starts and then change to

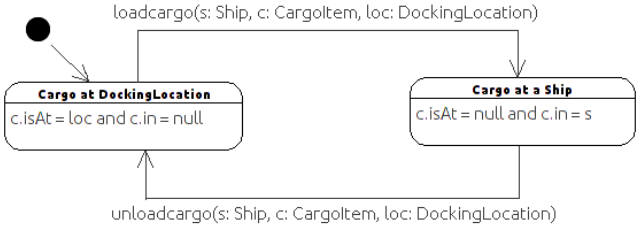


Figure 4: State machine diagram of the Cargo.

true when it ends. In itSIMPLE, we can represent this effect in the timing diagrams or in the OCL conditions. For example, *readytonavigate* is used to control the status of the ship when navigating from one location to another, preventing the planner from assigning another navigation action during the operation. As an effect of action *navigatewithnocargo* for instance, *readytonavigate* must be set to false at the start and then set to true at the end (when the ship arrives at the destination), as done in PDDL.

If a timing diagram is not used, temporal operators are annotated to the OCL pre- and post-conditions. For example, the variable *availablespots* is decreased as soon as the action *dock* is started, preventing any other ship docking at the same spot. This is done by annotating the post-condition 'loc.availablespots = loc.availablespots - 1' to the interval [start,start]. Users can also specify numeric intervals; however, PDDL does not support such indexation of time points. Property *readytonavigate* is also set to false when *dock* starts, 's.readytonavigate = false' in the interval [start,end]. Undocking is similar, but the variable *availablespots* is increased at the end and *readytonavigate* is set to true at the end. Therefore, we can guarantee that navigation will not be assigned during the whole process of docking and undocking. Moreover, the refueling operation must guarantee that the ship remains docked for the entire duration of the action. That is done by annotating the precondition 's.docked = true' with the interval [start,end]. In PDDL,

this precondition would be be translated to '(over all (docked ?s))'.

In order to illustrate the resulting specification of the actions and facilitate their understanding, we present below the PDDL code for the actions *navigate2waitingarea* and *loadcargo*. This code was generated automatically by itSIMPLE.

```
(:durative-action navigate2waitingarea
 :parameters(?s - Ship ?from - Location ?to - WaitingArea
    ?next - DockingLocation)
 :duration
    (= ?duration (/ (distance ?from ?to) (speed ?s)))
 :condition
    (and (at start (at ?s ?from))
      (at start (readytonavigate ?s))
      (at start (canrefuel ?next))
      (at start (>= (currentfuel ?s)
        (+ (* (distance ?from ?to) (lowerfuelrate ?s))
        (* (distance ?to ?next) (lowerfuelrate ?s))))))
 :effect
    (and (at end (at ?s ?to))
      (at end (decrease (currentfuel ?s)
        (* (distance ?from ?to) (lowerfuelrate ?s))))
      (at end (increase (totalfuelused)
        (* (distance ?from ?to) (lowerfuelrate ?s))))
      (at end (not (at ?s ?from)))
      (at start (not (readytonavigate ?s)))
      (at end (readytonavigate ?s))))

(:durative-action loadcargo
 :parameters (?s - Ship ?c - CargoItem
    ?loc - DockingLocation)
 :duration (= ?duration (/ (weight ?c) (loadingrate)))
 :condition
    (and (at start (at ?s ?loc))
      (at start (docked ?s))
      (at start (>= (loadcapacity ?s)
        (+ (currentload ?s) (weight ?c))))
      (at start (isAt ?c ?loc))
      (at start (craneidle ?s)))
 :effect
    (and
      (at end (increase (currentload ?s) (weight ?c)))
      (at end (in ?c ?s))
      (at end (not (isAt ?c ?loc)))
      (at start (not (craneidle ?s)))
      (at end (craneidle ?s))))
```

In itSIMPLE, UML object diagrams are used to describe the initial state and the goal state of a planning problem instance. The object diagram represents a picture of the system in a specific state. It can also be seen as an instantiation of the domain structure defined in the class diagram. This instantiation defines four main aspects: the number and type of objects in the problem; the values of the attributes of each object; and the relationships between the objects. In our problem, the initial state consists of a set of ships at their corresponding waiting areas and with the corresponding property values, the cargo items and their respective initial locations (ports), the platforms with their available spots and refueling capability, as well as all the distances between the existing location objects (this information can be inserted by importing data in a text file as opposed to manually inputting the information). The goal state is an object diagram in which all cargo items are at their destination and the ships are back to their respective waiting areas.

Besides the object diagrams for defining initial and goal states, we also model the objective function to be optimized in every planning situation. In itSIMPLE, we select the domain variable to be minimized in a way that allows it to be represented as a linear function in the *:metric* section of PDDL. In this model we consider (1) the total fuel used (stored in the variable *totalfuelused*) and (2) the makespan. The cost of docking time of each ship is not considered in this work due to limitation on available general planners in dealing with continuous properties/time. The continuous approach could be used to compute the time that a ship remains docked for its operations, providing the necessary costs to be considered during planning.

Model Testing with Planners and Plan Analysis

itSIMPLE can automatically generate a PDDL model from a UML representation. In addition to the automated translation process, the tool can communicate with several planners in order to test the domain models in an integrated design environment. In this application, the planners must be selected based on the resulting PDDL model requirements that extend beyond the classical approaches.

In order to analyze the generated plans, itSIMPLE provides two main support tools for plan analysis: simulation and validation. Plan simulation is performed by observing a sequence of snapshots (UML object diagrams), state by state, generated by applying the plan from the initial state to the goal state. The tool highlights every change in each state transition as described by Vaquero et al. (2007). For the plan analysis, itSIMPLE provides charts that represent the evolution of selected variables such as those related to the quality of a plan (metrics). In addition, itSIMPLE provides the use of the tool VAL[1] to validate the plans generated by PDDL-based planners.

Experimental Results

We present two case studies in this section to demonstrate how planners solve the ship operations problem (in one scenario for expansion of the platforms network) using the model generated by itSIMPLE. In the first case study, we investigate the performance of three classical, modern planners using a reduced version of the model in which no time constraints are considered. We focus on plan feasibility and the minimization of fuel consumption. Time constraints usually add more difficulty to the AI P&S techniques so we aim to set up a baseline performance with such a first study. In the second case study, we analyze the output of three modern planners using the model described in the paper, i.e., with the time constraints and requirements; however, only the makespan is considered in the minimization function. In this latter study, we selected three planners that were able to read and correctly handle the PDDL durative-actions present in the model.

[1] Available at http://planning.cis.strath.ac.uk/VAL/

In both case studies, we investigate different delivery request scenarios. We analyze the performance of the selected planners in problem instances with the number of cargo items equal to: 5, 7, 9, 11, 13, and 15 (with different weights). In these instances, *P1* has n cargo items while *P2* has $n + 1$ to simulate unbalanced requests. The problem instance with 15 cargo items represents a realistic demand from the platforms. In all instances, there are six ships in *A1* and four in *A2* in the initial state–all of them with 600 liters of fuel capacity, 400 liters of fuel, 100 tons of load capacity, no cargo, an average speed of 70 km/h, 0.3 l/km and 0.2 l/km as the higher and lower fuel consumption rates, respectively. In addition to the ports, platforms *F5* and *G3* are able to perform refueling. 100 l/h is the refueling rate at the ports and at platforms *F5* and *G3*. Docking and undocking durations are set to 1 hour in the ports and 0.5 hour in the platforms.

In our experiment, planners were run on an Intel Core i7 950 3.07 GHz computer with 4.00 Gigabytes of RAM.

Case Study 1: No Time Constraints

In this case study we consider a simplified model with no time constraints. Taking into account the model in Figure 2, we do not include the variables related to time and rates such as *loadingrate*, *speed*, *refuelingrate* and *dockingduration*. Actions are adapted accordingly. In fact, they are used only in the definition of action durations.

We selected the planners Metric-FF (Hoffmann 2003), SGPlan6 (Hsu and Wah 2008) and MIPS-xxl 2008 (Edelkamp and Jabbar 2008) for this experiment. Other planners such as LPG, LPG-td, LPRPG were also tried for this experiment but they could not handle the model (e.g., the planner halts with a segmentation fault). We investigate the performance of Metric-FF and MIPS-xxl 2008 with and without the optimization flag on. To analyze the planners' performance we look at the generated plans from the six problem instances (*p05, p07, p09, p11, p13, p15*) and measure the runtime, number of actions in the plan and the total fuel used by ships. We assigned a 6-hour timeout for the planners. Table 4 shows the results from this case study.

As shown in Table 4, Metric-FF without optimization is able to provide a solution to every problem instance. However, the planner is unable to solve any problems with the optimization flag on[2] – the time limit is reached in every case. SGPlan6 is not able to solve problems *p09* and *p15*: the planner stopped before reaching the time limit. Nevertheless, SGPlan6 outperforms Metric-FF in *p07* and *p11*in terms of the number of actions and the total fuel used. Metric-FF outperforms SGPlan6 in most of the cases. MIPS-xxl 2008 in terms of the number in all problem instances.

Analyzing the plans generated by Metric-FF without optimization, we detected that even though several vessels are available for the operations, the planners provide solutions in which just a few ships are used. For example, in the plan generated for the problem *p05*, only one ship (S9) is used for all deliveries and transportations. Only three ships (S7,S8,S9) are used to solve the problem *p15*. In addition, some plans contained unnecessary consumption of fuel, for example in cases where a ship travels from location A to B and then from B to C without doing any delivery, while it could go directly from A to C using less fuel (shorter distance). SGPlan6 shows a similar behavior by using a few ships to solve the problems; however, it does not show the unnecessary fuel consumption behavior.

[2]Since Metric-FF is treated as a blackbox in this experiment, we did not explore the reasons for why it does not solve any of the problems.

Case Study 2: With Time Constraints

In this case study we consider the complete model of ship operations in the port and platforms, with time constraints and requirements, illustrated in Figure 2. We selected planners POPF (Coles et al. 2010), SGPlan6 and MIPS-xxl 2008 for this experiment. POPF participated in the seventh International Planning Competition (2011) in the deterministic, temporal satisficing track. We have set up POPF to generate as many solutions as it could in the time limit, improving the plan quality (makespan in this case) in each subsequent solution. Other planners such as LPG-td and LPRPG were also tried for this experiment but they could not handle the model. To analyze the selected planners' performance we looked at the generated plans from the six problems instances (*p05, p07, p09, p11, p13, p15*) and measured the runtime, number of actions in the plan, the total fuel used by ships and the makespan. We assigned a 3-hour timeout for the planners to simulate a more realistic response horizon. Table 5 shows the results for the second case study.

As shown in Table 5, SGPlan6 is the only planner in this experiment that managed to solve some of the instances. Surprisingly, the more recent planner POPF does not solve any of the problem instances. We have also checked smaller problems with 2 and 3 cargo items, and even 1 cargo item and 1 ship, but it still does not solve them. SGPlan6 produced exactly the same solutions as in case study 1; the runtimes were greater in most of the cases though.

Discussion

The case studies showed that an AI P&S approach for solving the ship operation problem in Petrobras is possible; however, the available domain-independent planners do not currently provide the necessary set of tools to solve the modeled problem in real life. SGPlan6 can often provide a feasible solution, but optimal solutions in a realistic horizon do not appear to be achievable. As opposed to modeling the problem using optimization approaches (e.g., using MIP or CP models), our intention was to develop a model in order to evaluate if current planners would have acceptable performance in real scenarios. From the results presented in the previous section, we conclude that the planners do not succeed at this task.

Since one of the main goals in this paper is to describe the modeling experience, in this investigation we tried to model the problem using KE tools that would direct the model to standard representation languages in AI P&S and therefore could potentially be read by several planners. In fact, modeling the problem in UML and then translating to PDDL was

Cargo	Metric-FF						SGPlan6			MIPS-xxl 2008		
	no optimization			with optimization			With Metric			With and without optimization		
	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)	Runtime (s)	# actions	Fuel (l)
5	13.56	42	631	Timeout	-	-	12.11	47	781	Timeout	-	-
7	38.50	59	881	Timeout	-	-	933.68	58	805	Timeout	-	-
9	106.94	76	1,073	Timeout	-	-	X	-	-	Timeout	-	-
11	244.45	88	1,523	Timeout	-	-	1,114.42	96	1457	Timeout	-	-
13	284.68	105	1,533	Timeout	-	-	1,041.22	108	1630	Timeout	-	-
15	499.47	122	1,844	Timeout	-	-	X	-	-	Timeout	-	-

Table 4: Results from Case Study 1 - No Time Constraints. 'Timeout' means that the planner reached the 6-hour limit without generating any plan. 'X' means that the planner stopped before reaching the timeout limit without generating a plan.

Cargo	POPF				SGPlan6				MIPS-xxl 2008			
	Runtime (s)	# actions	Fuel (l)	Makespan (h)	Runtime (s)	# actions	Fuel (l)	Makespan (h)	Runtime (s)	# actions	Fuel (l)	Makespan (h)
5	X	-	-	-	1.82	47	781	152.52	Timeout	-	-	-
7	X	-	-	-	3,071.61	58	805	294.34	Timeout	-	-	-
9	X	-	-	-	X	-	-	-	Timeout	-	-	-
11	X	-	-	-	3,245.85	96	1,457	422.24	Timeout	-	-	-
13	X	-	-	-	1,180.81	108	1,630	600.54	Timeout	-	-	-
15	X	-	-	-	X	-	-	-	Timeout	-	-	-

Table 5: Results from Case Study 2 - With Time Constraints. 'Timeout' means that the planner reached the 3-hour limit without generating any plan. 'X' means that the planner stopped before reaching the timeout limit without generating a plan.

feasible and practical. The semantics of the model results in a natural mapping between real objects and objects in the model. Moreover, the mapping of the generated solution follows the same rules and has a direct map to the real world. This modeling ease is not necessarily true in the models developed with optimization technology.

It is indeed possible to refine and adapt the model so that planners could run faster and produce better solutions. A designer could even reduce the problem to a basic form so other planners can handle it. However, we tried to perform the modeling process by focusing on the semantics of the model – keeping the mapping obvious for non-planning experts. In fact, the resulting model can be seen as a transportation problem (the class of problems addressed the most by the AI Planning community) with extensions that make it more realistic (e.g., load capacity, fuel capacity). The model does not seem to be far different from what we see in classical numeric and temporal domains (e.g., logistics, depots, driverlog, zenotravel, etc.), but it indeed combines certain requirements that test the limits of the state-of-the-art planners. Therefore, it is a challenge domain for AI P&S approach. That is why it has been proposed as one of the challenge domains in the ICKEPS'12 competition.

Conclusion

In this paper, we have investigated a real planning problem, the planning and scheduling of ship operations in ports and platforms, using an AI P&S approach. We described the design process used for building a domain model with the KE tool itSIMPLE. In order to validate the model and investigate the applicability of state-of-the-art planners in this problem, two case studies were conducted. The first one considers a semi-realistic scenario in which no time constraints are considered and the second brings a more realistic case in which time is considered. The planners were selected based on their capacity in dealing with the domain model requirements (durative-actions, numeric variables, and metrics). The metrics considered in these problems focus on the minimization of different parameters such as total fuel used by ships and the makespan.

Experimental results showed that in both cases some planners can provide valid solutions for the problem, however, they struggle to provide solutions to more realistic problems. It is important to note that few planners can deal with such a combination of PDDL features. Therefore, the resulting PDDL model brings interesting challenges even for the state-of-the-art planners. The model will be made available in order to share our results on this domain. In addition, experience from this application has motivated the improvement of itSIMPLE towards time-based models to support designers on real-world problems.

Acknowledgment

The first author is supported by the Government of Canada Post-Doctoral Research Fellowship. The second author is supported by FAPEAM.

References

Coles, A. J.; Coles, A. I.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Edelkamp, S., and Jabbar, S. 2008. MIPS-XXL: Featuring External Shortest Path Search for Sequential Optimal Plans and External Branch-And-Bound for Optimal Net Benefit. In *Short paper for the International Planning Competition 2008*.

Hoffmann, J. 2003. The metric-FF planning system: Translating ignoring delete lists to numerical state variables. *Journal of Artificial Intelligence Research (JAIR)* 20.

Hsu, C.-W., and Wah, B. W. 2008. The sgplan planning system in ipc-6. In *Proceedings of the Sixth Internation Planning Competition (IPC) in ICAPS 2008*.

Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *In Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 526–533. AAAI Press.

Kautz, H., and Walser, J. P. 2000. Integer optimization models of ai planning problems. *The Knowledge Engineering Review* 15:2000.

OMG. 2003. *UML 2.0 OCL Specification m Version 2.0*.

OMG. 2005. *OMG Unified Modeling Language Specification, m Version 2.0*.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An integrated tool for designing planning environments. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007). Providence, Rhode Island, USA*.

Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F.; and Beck, J. C. 2009. From requirements and analysis to PDDL in itSIMPLE3.0. In *Proceedings of the Third ICKEPS, ICAPS 2009, Thessaloniki, Greece*.

Vaquero, T. S.; Silva, J. R.; and Beck, J. C. 2011. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the ICAPS 2011 workshop on Knowledge Engineering for Planning and Scheduling workshop. Toronto, Canada*.

Constraint-based Scheduling for Closed-loop Production Control in RMSs

E. Carpanzano & A. Orlandini & A. Valente
ITIA-CNR
Italian National Research Council
Milan, Italy

A. Cesta & F. Marinò & R. Rasconi
ISTC-CNR
Italian National Research Council
Rome, Italy

Abstract

Reconfigurable manufacturing systems (RMS) are conceived to operate in dynamic production contexts often characterized by fluctuations in demand, discovery or invention of new technologies, changes in part geometry, variances in raw material requirements. With specific focus on the RMS production aspects, the scheduling problem implies the capability of developing plans that can be easily and efficiently adjusted and regenerated once a production or system change occurs. The authors present a constraint-based online scheduling controller for RMS whose main advantage is its capability of dynamically interpreting and adapting to production anomalies or system misbehavior by regenerating on-line a new schedule. The performance of the controller has been tested by running a set of closed-loop experiments based on a real-world industrial case study. Results demonstrate that automatically synthesizing plans and recovery actions positively contribute to ensure a higher production rate.

Introduction

Highly automated production systems are devised to efficiently operate in dynamic production environments, as they implement at various levels the capability to adapt or anticipate uncertainty in production requirements (Smith and Waterman 1981; Wiendahl et al. 2007). Generally, Reconfigurable Manufacturing Systems (RMS) are endowed with a set of reconfigurability enablers related either to the single system component (e.g., mechatronic device, spindle axes), or related to the entire production cell and the system layout; as a consequence, possible fluctuations of the production demand can be counteracted by implementing the required enablers. Differently from RMSs, in Focused Flexibility Manufacturing Systems (FFMS) the responsiveness towards the changes relies on the production evolution forecasting. On the basis of the predicted events, the production system is preliminarily endowed with the necessary degree of flexibility which is exploited at the moment the change occurs (Terkaj, Tolio, and Valente 2009; 2010).

A particularly interesting case concerns the integration of production and automation RMS layers, as failing to provide an efficient integration between the previous two modules may severely affect the system global performance (Valente and Carpanzano 2011; Carpanzano et al. 2011). A production schedule module designed for highly automated systems must be able to manage both exogenous (e.g. change of volumes or machining features) and endogenous events (e.g. machine failures or anomalous behavior). At the same time, it must close the loop with the automation dispatching module, which is responsible for mapping production tasks into the related automation tasks that are assigned to the devices, coherently to the scheduled production jobs sequences. Closing the loop between the two modules entails that the dispatching module continuously feeds back the current status to the production schedule module, which may decide to possibly modify the plan (Fig. 1).

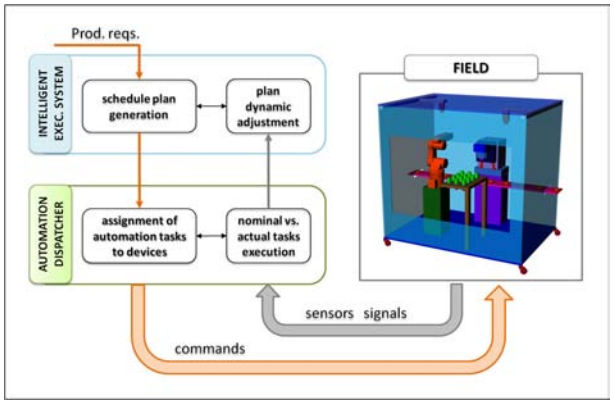


Figure 1: Production scheduler and automation dispatcher closed-loop.

There is a number of production scheduling approaches considering changes, both static (Tolio and Urgo 2007) and dynamic (Rasconi, Policella, and Cesta 2006). Another similar example of deployment of Planning & Scheduling techniques for on-line planning and execution in real-world domains can be found in (Ruml et al. 2011), where the authors tackle the problem of controlling production printing equipment by exploiting an on-line algorithm combining state-space planning and partial-order scheduling to synthesize plans. As opposed to (Ruml et al. 2011), the emphasis in the work presented here is more focused on the exploitation of the plan's temporal flexibility during the execution phase to hedge against the environmental uncertainty. More in detail,

while in (Ruml et al. 2011) the main effort revolves around the on-line planning, makespan-optimization and dispatching of each new printing requests (goals) with plan abortion in case of a printer module failure, in our work great attention is devoted to the on-line plan readjustment in case exogenous events occur during execution. In our case, less effort is devoted to planning, as a determined sequence of tasks is provided for each different production request (i.e., there is no need to *plan*, in the classical sense); rather, we focus on the production plan AI-based scheduling followed by the on-line rescheduling and/or corrective temporal propagation, should disruptions make the plan resource-unfeasible at execution time.

With specific focus on the RMS management aspects, the production scheduling problem implies the capability to develop a short term production plan based on the inputs generated by the capacity planning problem that can be easily and efficiently adjusted and regenerated once a production or system change occurs. Despite the capability of generating robust and adaptive scheduling plans, the available approaches described above are decoupled by the system automation layer. The work addressed in this paper attempts to fill this gap, by merging the production and automation scheduling modules in a RMS context, and presenting the system resulting from this integration applied to a real industrial case. The paper is structured as follows: after presenting the proposed dynamic production scheduling approach, we analyze a particular case study taken from an industrial application; we then proceed to describe the formulation of the scheduling model, and finally we outline the major benefits of the approach, closing the paper with some final observations about the ongoing work.

The proposed approach

In (Carpanzano et al. 2011) we proposed to address the production scheduling problem using the Constraint Satisfaction Problem (CSP) formalism, as it allows to naturally express the features needed to model scheduling problems under uncertainty (Rasconi, Policella, and Cesta 2006) (e.g., it allows to easily provide the search algorithms with domain-specific heuristic, and to naturally represent *flexible* solutions). This characteristics provide the schedule with strong reconfiguration capabilities during execution, should potentially disrupting events occur. Synthesizing a production plan basically entails assigning the available resources to the jobs that are to be processed in the plant with a temporal horizon of the shift; once jobs are allocated to the resources, the schedule is passed to the automation layer that translates the production scheduling in automation plans.

Modeling the scheduling features

The base scheduling problem model employed in this work conforms to the *Resource Constrained Project Scheduling Problem with Time Lags* (RCPSP/max), this is to open the possibility to import a robust algorithmic experience on the problem (Cesta, Oddi, and Smith 2002; Rasconi, Policella, and Cesta 2006). The RCPSP/max can be formalized as follows: (**i**) a set V of n activities must be executed, where each activity a_j has a fixed duration d_j. Each activity has a start-time S_j and a completion-time C_j that satisfies the constraint $S_j + d_j = C_j$; (**ii**) a set E of temporal constraints exists between various activity pairs $< a_i, a_j >$ of the form $S_j - S_i \in [T_{ij}^{min}, T_{ij}^{max}]$, called start-to-start constraints (time lags or generalized precedence relations between activities); (**iii**) a set R of renewable resources are available, where each resource r_k has a integer capacity $c_k \geq 1$. The execution of an activity a_j requires capacity from one or more resources; for each resource r_k the integer $rc_{j,k}$ represents the required capacity (or size) of activity a_j. A schedule S is said to be *time-feasible* if all temporal constraints are satisfied, while it is *resource-feasible* if all resource constraints are satisfied (let $A(S,t) = i \in V | S_i \leq t < S_i + d_i$ be the set of activities which are in progress at time t and $r_k(S,t) = \sum_{j \in A(S,t)} rc_{j,k}$ the usage of resource r_k at that same time; for each t the constraint $r_k(S,t) \leq c_k$ must hold). The solving process is performed exploiting a makespan optimization scheduling algorithm called ISES (Iterative Sampling Earliest Solutions) (Cesta, Oddi, and Smith 2002) . The ISES solving algorithm basically proceeds by detecting the sets of schedule activities that compete for the same resource beyond the resource maximum capacity (*conflict sets*) and deciding the order of the activities in each set, through the insertion of further temporal constraints between the end time of one activity and the start time of the other, to eliminate conflicting overlaps.

The Dynamic Scheduling Control Architecture

In this work, we present a real-time control architecture (see Fig. 2) endowed with the flexible production scheduling capabilities discussed above in order to dynamically synthesize updated scheduling solutions as required by the continuously changing environmental conditions.

As shown in Fig. 2, the control architecture is designed to provide/receive data to/from the automation layer, and is composed of three different modules, each one holding different responsibilities. The *Controller* is the main component of the architecture and is in charge of: (**i**) invoking the Scheduler in order to ask for new solutions whenever a new job is entering the system (*find solution* command, see also the following point iv); (**ii**) updating the internal model of the system according to the observations received by the Dispatcher (*modify model* command); (**iii**) detecting any possible cause (e.g., anomalous behaviors, failures, etc.) leading to plan unfeasibility; (**iv**) invoking the Scheduler in order to reschedule the current solution and possibly produce a new feasible solution; (**v**) disposing completed tasks from the current model. Whenever invoked by the Controller, the *Scheduler* is responsible for (**i**) producing the initial solution needed to initiate the production process starting from a given problem, and (**ii**) rescheduling the current solution when it becomes unfeasible due to the onset of some exogenous event. Finally, the *Dispatcher* is responsible for (**i**) realizing the communication from the automation level to the rest of the architecture (all messages coming from the field are pre-processed by the Dispatcher and the related data are forwarded to the Controller), and (**ii**) dis-

patching solution-related plan activation signals to the automation layer.

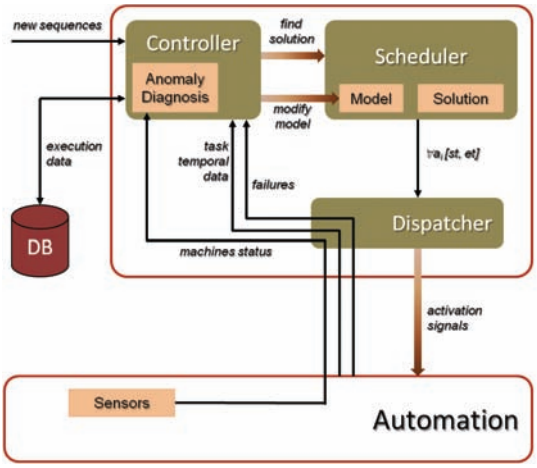


Figure 2: The Overall Control Architecture.

The overall architecture is implemented in Java as a composition of three concurrent and asynchronous processes that interact in a coordinated way to control the production process. In addition, one additional component has been implemented in order to record and store in a database the information flowing within the control system and to provide a human operator with a graphical view of the collected data. Finally, the communication between the control architecture and the automation level has been implemented through the use of the OPC protocol. According to the ISA95 standard, such protocol is fully compatible for SCADA connection.

Representing Maintenances and Recovery actions

In order to make the execution domain as close as possible to the real production system environments, besides the ordinary production tasks the system is able to accommodate *maintenance* activities (ordinary and extraordinary) as well as *recovery* actions that should be executed after a machine failure. Ordinary maintenances are generally scheduled in the plan according to their due frequency, extraordinary maintenances are scheduled in case of anomalous machine behaviors, while recovery actions are instead inserted in the plan on occurrence of particular machine failures. The urgency (i.e., the execution immediacy) of the extra-maintenance will be decided on the basis of the gravity of the occurred anomaly, which is assessed by the Controller's *Anomaly Diagnosis* module (see Fig. 2). It should be noted that as opposed to anomalies (which entail a degraded machine performance), we assume failures entail the complete inoperability of the affected resource until the failure is resolved (see Section *Production and management features of the FRC* for details related to the use case considered in this work).

Industrial case application

The proposed scheduling approach has been applied to an industrial case pertaining to a reconfigurable production line for the manufacturing of customized shoes, representing the European Best Practice in mass customization. The production system is composed by 5 manufacturing cells connected by a flexible transport system composed by rotating tables. The last automated manufacturing island in the shop-floor (Fig. 3) is the Finishing Robotic Cell (FRC), responsible for the shoe finishing before packaging and delivery.

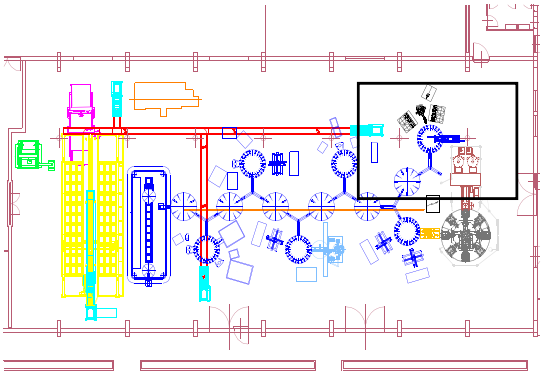


Figure 3: Shop-Floor Layout and Finishing Robotic Cell location.

As illustrated in Fig. 4, the FRC consists of four machine units, respectively an ABB robot (R1), the island from/to which parts are un/loaded (R2), a controlled brushing machine (R3), a creaming machine (R4) and a spraying machine (R5). The robot operates as *pick and place* and fixturing system; it loads the semi-finished shoe from the island (or rotary table) and, according to the part program, transports the part to the related machines, holding the part while the machine is processing it, as a proper fixturing system. Creaming and spraying machines are equipped with two inter-operational buffers with 9 slots each.

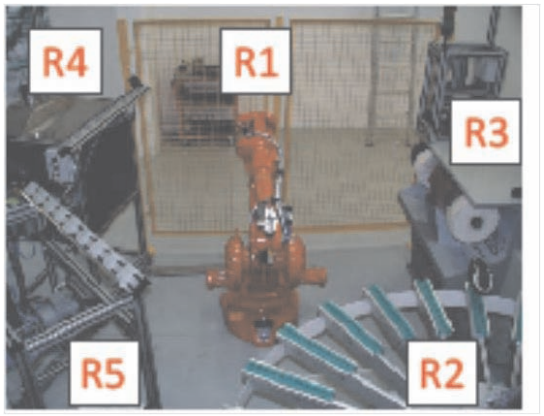


Figure 4: Resource composing the Finishing Robotic Cell.

As far as the FRC automated system is concerned, the FRC controller is connected with the transportation system PLC, the SCADA of the entire line and the low lever cell controller modules. Three types of activities are achieved by means of the existing control architecture: Communication-synchronization with production line controller; Synchronization of tasks in the finishing cell; Control of finishing operations such as rotation speed of the felt rollers, check of

spray pressure and drying time, tracking of actual operation execution times compared to nominal expected ones.

Production and management features of the FRC

The FRC finishing process can be clustered in three main families: creaming processes, spraying processes and brushing processes. A typical process sequence is structured in the following steps: part loading; brushing for cleaning the raw piece of dust; finishing by spraying or creaming operations; drying in the buffer; brushing; unloading the finished part.

As highlighted in (Carpanzano et al. 2011), the considered family of products consists of 8 different part types (i.e., 4 woman models and 4 male models). The processing of each part is to be further divided into the left and right subparts of each shoe model. The production of all parts can be described in terms of the task sequences presented in Table 1. Given a specific shoe model, the left and right part of the

Table 1: Description of operation sequences.

Sequence #1	**Sequence** #2	**Sequence** #3	**Sequence** #4
Load	Load	Load	Load
Brushing	Brushing	Spraying	Creaming
Spraying	Creaming	Unload	Unload
Unload	Unload	Buffering	Buffering
Buffering	Buffering	Load	Load
Load	Load	Brushing	Brushing
Brushing	Brushing	Unload	Unload
Unload	Unload		

model can be produced by means of the same sequence type for both female and male items. However, the durations of the sequence tasks can vary depending on the product type, resulting in 16 different process sequences in total.

As stated earlier, besides the production tasks a number of maintenance operations need to be foreseen and scheduled to ensure the FRC health. Table 2 synthesizes a few examples of maintenance tasks for FRC resources, considered in this work; in the table, the listed maintenance activities are associated to the related resource, and it is specified whether a stop of the cell is required. The table reports the average expected time (in seconds) for carrying out each maintenance activity as well as the maintenance rate indicated in brackets.

Table 2: Maintenance Operation Time matrix [sec].

Maint. Task [Rate]	**R1**	**R2**	**R3**	**R4**	**R5**	**Stop**	**Fqncy**
Fill cream tank					90	no	1/day
Creaming M. Clean.					60	no	12/day
Creaming M. Nozzle Clean.					3	no	2/hour
Fill spray tank				60		no	1/day
Spraying M. Clean.				60		no	12/day
Spraying M. Nozzle Clean.				3		no	2/hour
Fill wax in Brushing M.			60			no	1/day
Gripper Calibr.	15					no	1/day

Besides the maintenance tasks, a set of FRC failures have also been systemized and clustered by type in this work (see Table 3). Each failure type mapped upon resources is associated to a number of suitable troubleshooting strategies. An efficient execution of maintenance and/or recovery tasks relies on a persistent signal interpretation to assess the system status. This evaluation is crucial to identify the gap between actual and nominal system behavior and consequently the related actions to be implemented. Table 4 outlines few examples of signal information associated to the need to undertake specific maintenance tasks. For each considered machine maintenance, the table shows: (**i**) the polled sensors, and (**ii**) the predefined signal threshold values beyond which anomalies of different gravity are recognized (e.g., severe (*red*) anomalies are detected when the weighted sum of the anomalous readings obtained from sensors goes below 10%).

Table 3: Failure modes.

Fail. types	**R3**	**R4**	**R5**	**Dur. (mins)**	**Cell Stop**
Wax not moving	x			2	no
Brush slider not moving	x			2	no
Brush not rotating	x			2	no
Dosage not working			x	5,15	no,yes
Cream not arising from sponge			x	15,25	no,yes
Spray pistol not responding		x		10,20	no,yes
Air only from spray pistol		x		5	no
Anomalous spray pistol jet		x		10	no

Table 4: Maintenance tasks from signal interpreting.

Maintenance type	**Sensors**	**Orange**	**Red**
Fill cream tank	Level	10-20%	0-10%
Fill spray tank	Level	10-20%	0-10%
Fill wax in Brushing M.	Level	10-20%	0-10%
Gripper calibration	Force sensor	10-20%	0-10%
Creaming M. cleaning	Visual + filter + prod. qlty	15-30%	0-15%
Spraying M. cleaning	Visual + filter + prod. qlty	15-30%	0-15%
Creaming M. nozzle clean.	Cream cons. + valve + prod. qlty	15-30%	0-15%
Spraying M. nozzle clean.	Spray cons. + valve + prod. qlty	15-30%	0-15%

The scheduling-based controller

As explained in (Carpanzano et al. 2011), the FRC scheduling problem is modeled in CSP terms adopting a combination of modeling strategies that allows to capture all the significant aspects of the problem that the solving process must reason upon.

Modeling in the static case

The reader interested in the base model details can refer to (Carpanzano et al. 2011); in that work, we focused on a *model abstraction* suitable for the *static* problem solving case, which has allowed us to: (1) decrease the number of involved tasks guaranteeing no loss of expressiveness, and (2) re-use partially modified, if at all, off-the-shelf scheduling algorithms for the solving process.

The solution provided in (Carpanzano et al. 2011) was taking advantage of the robot acting as a *critical* resource, which allowed the two task subsequences immediately preceding and following the buffering operation to be grouped in two single blocks (the first and the third dashed boxes, in Fig. 5). In order to allow for a finer treatment of machine faults and maintenance operations, in the present work it is necessary to abandon such *aggregated* model and keep

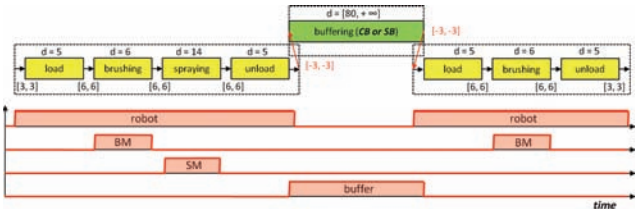


Figure 5: FRC task sequence for the woman #1 shoe part.

each individual sequence task separated. Fig. 5 depicts a typical sequence that entails the utilization of a subset of FRC machines and tools, e.g., the brushing machine and the spraying machine, as well as one of the two available buffers. Each sequence task is characterized by a nominal duration d, and consecutive tasks are separated by temporal constraints $[a, b]$ where a and b are the lower and the upper bound of the separation constraint. The actual constraint values depicted in Fig. 5 are consistent with the real robot transition times (e.g., the 6 value between the brushing and the spraying tasks represents the time that the robot takes to go from the brushing machine to the spraying machine passing through the *home* position), while the negative constraint values shown in red characterize the fact that the buffering operation actually starts 3 seconds *in advance* with respect to the end of the first dashed box, because the robot must however return to its home position before commencing any other action.

The Dynamic Model

Interleaving deliberation and execution in a smooth and effective way is a crucial issue for real time model-based control systems. In particular, integrating deliberative and reactive control is not a straightforward task and, then, suitable mechanisms are needed in order to guarantee a robust and continuous control.

In literature, several solutions have been proposed. For instance, in (Lemaitre and Verfaillie 2007), the authors propose a generic schema for the interaction between reactive and deliberative tasks where reactive and high-level reasoning control tasks are implemented and integrated so as to respectively meet a synchronous behavior assumption (i.e., in case of an exogenous event, a reactive task is always ready to be executed *before* any other event arrives), and an anytime behavior (i.e., a deliberative task is able to produce a first solution quickly, which can be improved later if time allows). Another approach is the one proposed in (Py, Rajan, and McGann 2010) where a hierarchy of reactors is exploited constituting several concurrent sense-plan-act control loops with different deliberation latencies. Both deliberative and reactive controls are implemented by means of, respectively, higher and lower latency reactors. In particular, reactors with small latencies are in charge to quickly react to unexpected events while reactors with long-term goals are managed by reactors with larger latencies.

In our case, given the chosen system latency and the FRC's characteristics, during the rescheduling phases the proposed control architecture is designed so as to (**i**) collect unexpected events (e.g., detected delays) that may occur during the rescheduling phases, and (**ii**) propagate such delays on the new solution generated for execution, by exploiting the solution's temporal flexibility. Such propagations/adjustments are guaranteed to be within the system latency by keeping the number of activities in the current schedule as low as possible, i.e., by eliminating the activities from the plan as they terminate their execution, in order to establish a sort of *dynamic equilibrium* between incoming and outgoing sequences, after an initial transient.

In order to allow the management of the schedule in a dynamic context (i.e., continuously absorbing all the modifications that pertain to the occurrence of exogenous events as well as to the simple passing of time) it has been necessary to extend the model presented in the previous section with online knowledge-capturing and management features. In our framework, such features are added using an asynchronous event-based model. All the information about the environmental uncertainty (e.g., endogenous and/or exogenous events) is organized through an asynchronous message exchange mechanism among the system modules. These messages convey all the information relatively to the deviations between the nominal schedule currently under execution and the *real data* coming from the automation side of the plant. The Controller (see Fig. 2) is in charge of acquiring such information, adapting the plan accordingly, and calling for the necessary rescheduling actions. In particular, a global rescheduling is performed each time a new sequence (i.e., a new production order) is inserted in the plan. However, applying a rescheduling to an executing plan generally presents the technical difficulty arising from the fact that the Scheduler does not have any internal chronological model of the schedule with respect to the passing of time. In other words, it has no knowledge of *past*, *present* and *future* relatively its own activities (i.e., it may decide to reschedule one activity into the past, or postpone the start time of an activity that has already started).

The latter issue is solved by introducing a number of constraint-based pre-processing procedures whose objective is to impose new constraints to the executing schedules prior to the solving process, so as to *force* the Scheduler to produce solutions that reflect the temporal reality of execution. Such procedures are the following: (**i**) *fixActivity()* when the Dispatcher acknowledges from the plant that an activity has started, the Controller must *fix* the activity's start time in the model, so that it is not shifted by the rescheduling process; (**ii**) *fixActivityDuration()* when the Dispatcher acknowledges from the plant that an activity has terminated, the Controller must fix the activity's end time, so that the latter is not modified by any possible rescheduling process before the activity is eliminated from the current plan; (**iii**) *disposeCompletedActivity()* this procedure eliminates a completed activity from the model; (**iv**) *prepareRescheduling()* this procedure performs the very important task of inserting in the plan a set of new *release constraints* relatively to all the activities that will participate to the rescheduling, so as to avoid that such activities will be scheduled in the past w.r.t. to the current execution time. Once all previous preparatory actions are performed, the rescheduling proce-

dure can be safely called by the Controller. The Scheduler will therefore produce an alternative solution that (**i**) is temporally and resource feasible, (**ii**) satisfies all problem-related and execution-related constraints, and (**iii**) complies with the chronological physical requirements.

Experimental Results

In this section, we analyze the dynamic scheduling performances of our architecture by deploying it to control the execution of a series of typical production tasks relatively to the FRC case study. In particular, we will test the dynamic scheduling capabilities of our system by simulating the execution of a determined number of production sequences, which entails the online scheduling of the continuously incoming production tasks (equally distributed among the different process types) and ordinary maintenances (defined in Tab. 2). Both the temporal flexibility of the employed model and the rescheduling efficacy of the solver will be assessed by simulating the onset of perturbing events of random extent during each execution. More specifically, we analyze the performances of our architecture by varying the following settings: (**i**) we consider randomly variable start and end times for each incoming task, which affects the overall stability of the solution and requires the controller to continuously invoke the scheduler in order to adjust the current solution; (**ii**) we introduce a number of anomalies on the basis of the values (described in Tab. 4) detected by the automation layer sensors, and processed by the Diagnosis module. Each time an anomaly is detected, the control architecture reacts by scheduling an extraordinary maintenance activity whose urgency depends on the severity of the anomaly (*orange, red*). Maintenance activities may even cause the complete stop of the cell, and affect in any case the overall makespan; (**iii**) according to Tab. 3, we consider a set of possible failures for each machine, that may occur during execution. In this cases, the control architecture is in charge of scheduling the proper recovery task aimed at restoring full machine operability. As for anomalies, failures may introduce idle production periods, thus reducing production capability.

The experiments are organized in two different settings, both entailing the execution of 130 uniformly distributed production sequences. In the $1^{st}Setting$, 5 runs are executed for each resource R_i of the FRC. Each run requires the dynamic scheduling of the continuously incoming production tasks, including the periodic maintenances. Temporal uncertainty is introduced by considering an average 10% randomic misalignment between the nominal (i.e., dispatched) and the real (i.e., acknowledged) start/end times of the production activities. Each run is characterized by the onset of a number of anomalies and failures that depends on the affected machine R_i: in particular, every brushing machine will undergo 5 anomalies and 3 failures, every creaming machine will undergo 3 anomalies and 2 failures, and every spraying machine will undergo 3 anomalies and 3 failures (such numbers are decided on the basis of the available maintenance and recovery operations for each machine as well as of their durations, as per Tables 2 and 3). In order to appreciate the benefits of a controller that allows the concurrent scheduling and execution of both maintenance and production tasks, a second experimental setting is developed ($2^{nd}Setting$) where all previous runs are performed anew under the assumption that each maintenance and each failure recovery action entails a full FRC cell stop. All runs are performed on a MacBook Pro with a 64-bit Intel Core i5 CPU (2.4GHz) and 4GB RAM. In the following, we illustrate the collected empirical results.

Table 5 summarizes the obtained results; the table is horizontally organized so as to provide the data related to every machine. In particular, for each machine row the table lists data obtained in the first and second experimental settings (first and second row) together with the plain value difference and related percentage (third row). For each setting, the table provides the average values obtained from the five runs executed on each machine of: (**i**) the final makespan (i.e., the completion time of all 130 production sequences), (**ii**) the overall average time spent in reschedulings, (**iii**) the total number of reschedulings.

Table 5: Results from the experimental runs.

	MK (mins)	**Resched. T. (mins)**	**# of Resched.**
Brushing Machine			
$1^{st}Setting$	251	27	129
$2^{nd}Setting$	269	30	157
Δ (Δ %)	18 (7.2%)	3 (11.1%)	28 (21.7%)
Spraying Machine			
$1^{st}Setting$	256	26	130
$2^{nd}Setting$	279	28	155
Δ (Δ %)	23 (9%)	2 (7.7%)	25 (19.2%)
Creaming Machine			
$1^{st}Setting$	250	25	127
$2^{nd}Setting$	278	28	154
Δ (Δ %)	28 (11.2%)	3 (12%)	27 (21.2%)

The obtained results show the advantage of deploying an online reasoner that allows to continue execution during maintenances and recovery actions. Regardless of the machine involved in the performed runs, a significant reduction in makespan can be observed between the two experimental settings, meaning that the cell succeeds in executing all sequences in less time. In the table, makespan gains ranging from 18 up to 28 minutes are observable, which represent a significant improvement when measured against a total run time of 4 hours. Such gains are more evident for the machines that are characterized by longer maintenance and recovery actions (i.e., spraying and creaming). In case of long maintenances or recoveries, the capability to continue the execution of the tasks already scheduled on the unaffected machines is of great importance. Another interesting aspect can be observed by analyzing the higher number of reschedulings necessary in the $2^{nd}Setting$ w.r.t. to $1^{st}Setting$ runs; the reason of this stems from the fact that in order to simulate the absence of the execution controller ($2^{nd}Setting$ runs) we have modeled the cell-blocking condition by considering all maintenances and recoveries as tasks that require the whole cell; this causes a resource conflict that has to be solved by means of a rescheduling each time a maintenance or a recovery must be executed. As a last

observation, the table also confirms that the chosen number of failures and anomalies injected during all runs for the different machines was well balanced, as the average total time spent for reschedulings is equally subdivided in all cases of the same type, despite the durations of the recoveries and maintenances varied significantly among the machines (see Tables 2 and 3), the reason being that the longer the recovery/maintenance operation, the higher the possibility of a rescheduling when it is added to the plan.

Conclusions

This work has presented an AI-based online scheduling controller capable of dynamically manage a production plan under execution in uncertain environmental conditions. The capabilities of the proposed scheduling controller have been tested with reference to a real-world industrial application case study. The series of closed-loop experimental tests concerning the execution of reality-inspired production plans (i.e., complete with regular maintenances, as well as random failures and anomalies), demonstrate that thanks to the adopted flexible model, the proposed controller enhances the current production system with the robustness necessary to face a subset of typical real-world production requirement evolutions. The current results confirm that the deployment of continuous rescheduling capabilities on a temporally flexible plan model positively contribute to the overall efficiency of the production plant, by allowing the execution of the planned number of jobs in less time. The authors work is currently ongoing with the further objectives of (**i**) improving the controller's rescheduling optimization capabilities in environments characterized by a higher number of tasks, and (**ii**) expanding the controller's uncertainty management capabilities to the whole actual set of FRC exogenous events, which represents a necessary step before commencing any experimentation on the real field.

Acknowlegments. The research presented in the current work has been partially funded under the Regional Project "CNR - Lombardy Region Agreement: Project 3". Cesta and Rasconi acknowledge the partial support of MIUR under the PRIN project 20089M932N (funds 2008).

References

Carpanzano, E.; Cesta, A.; Orlandini, A.; Rasconi, R.; and Valente, A. 2011. Closed-loop production and automation scheduling in RMSs. In *ETFA. International Conference on Emergent Technologies and Factory Automation*.

Cesta, A.; Oddi, A.; and Smith, S. 2002. A Constraint-based Method for Project Scheduling with Time Windows. *Journal of Heuristics* 8(1):109–136.

Lemaitre, M., and Verfaillie, G. 2007. Interaction between reactive and deliberative tasks for on-line decision-making. In *Proceedings of the ICAPS 3rd Workshop on Planning and Plan Execution for Real-World Systems*.

Py, F.; Rajan, K.; and McGann, C. 2010. A systematic agent framework for situated autonomous systems. In *AAMAS*, 583–590.

Rasconi, R.; Policella, N.; and Cesta, A. 2006. Fix the Schedule or Solve Again? Comparing Constraint-Based Approaches to Schedule Execution. In *COPLAS-06. Proceedings of the ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

Ruml, W.; Do, M. B.; Zhou, R.; and Fromherz, M. P. J. 2011. On-line planning and scheduling: An application to controlling modular printers. *J. Artif. Intell. Res. (JAIR)* 40:415–468.

Smith, T., and Waterman, M. 1981. Identification of Common Molecular Subsequences. *J. Mol. Biol.* 147:195–197.

Terkaj, W.; Tolio, T.; and Valente, A. 2009. Design of Focused Flexibility Manufacturing Systems (FFMSs). *Design of Flexible Production Systems - Methodologies and Tools* 137–190.

Terkaj, W.; Tolio, T.; and Valente, A. 2010. A Stochastic Programming Approach to support the Machine Tool Builder in Designing Focused Flexibility Manufacturing Systems – FFMSs. *International Journal of Manufacturing Research* 5(2):199–229.

Tolio, T., and Urgo, M. 2007. A Rolling Horizon Approach to Plan Outsourcing in Manufacturing-to-Order Environments Affected by Uncertainty. *CIRP Annals – Manufacturing Technology* 56(1):487–490.

Valente, A., and Carpanzano, E. 2011. Development of multi-level adaptive control and scheduling solutions for shop-floor automation in Reconfigurable Manufacturing Systems. *CIRP Annals - Manufacturing Technology* 60(1):449–452.

Wiendahl, H.-P.; ElMaraghy, H.; Nyhuis, P.; Zah, M.; Wiendahl, H.-H.; Duffie, N.; and Brieke, M. 2007. Changeable Manufacturing - Classification, Design and Operation. *CIRP Annals - Manufacturing Technology* 56(2):783–809.

Planning for perception and perceiving for decision: POMDP-like online target detection and recognition for autonomous UAVs

Caroline P. Carvalho Chanel[1,2], **Florent Teichteil-Königsbuch**[2], **Charles Lesire**[2]
[1]Université de Toulouse – ISAE – Institut Supérieur de l'Aéronautique et de l'Espace
[2]Onera – The french aerospace lab
2, avenue Edouard Belin
FR-31055 TOULOUSE

Abstract

This paper studies the use of POMDP-like techniques to tackle an online multi-target detection and recognition mission by an autonomous rotorcraft UAV. Such robotics missions are complex and too large to be solved off-line, and acquiring information about the environment is as important as achieving some symbolic goals. The POMDP model deals in a single framework with both perception actions (controlling the camera's view angle), and mission actions (moving between zones and flight levels, landing) needed to achieve the goal of the mission, i.e. landing in a zone containing a car whose model is recognized as a desired target model with sufficient belief. We explain how we automatically learned the probabilistic observation POMDP model from statistical analysis of the image processing algorithm used on-board the UAV to analyze objects in the scene. We also present our "optimize-while-execute" framework, which drives a POMDP sub-planner to optimize and execute the POMDP policy in parallel under action duration constraints, reasoning about the future possible execution states of the robotic system. Finally, we present experimental results, which demonstrate that Artificial Intelligence techniques like POMDP planning can be successfully applied in order to automatically control perception and mission actions hand-in-hand for complex time-constrained UAV missions.

Introduction

Target detection and recognition by autonomous Unmanned Aerial Vehicules (UAVs) is an active field of research (Wang et al. 2012), due to the increasing deployment of UAV systems in civil and military missions. In such missions, the high-level decision strategy of UAVs is usually given as a hand-written rule (e.g. fly to a given zone, land, take image, etc.), that depends on stochastic events (e.g. target detected in a given zone, target recognized, etc.) that may arise when executing the decision rule. Because of the high complexity of automatically constructing decision rules, called policy, under uncertainty (Littman, Cassandra, and Pack Kaelbling 1995; Sabbadin, Lang, and Ravoanjanahary 2007), few deployed UAV systems rely on automatically-constructed and optimized policies.

When uncertainties in the environment come from imperfect action execution or environment observation, high-level policies can be automatically generated and optimized using Partially Observable Markov Decision Processes (POMDPs) (Smallwood and Sondik 1973). This model has been successfully implemented in ground robotics (Candido and Hutchinson 2011; Spaan 2008), and even in aerial robotics (Miller, Harris, and Chong 2009; Schesvold et al. 2003; Bai et al. 2011). Yet, in these applications, at least for the UAV ones, the POMDP problem is assumed to be available before the mission begins, allowing designers to have plenty of time to optimize the UAV policy off-line.

However, in a target detection and recognition mission (Wang et al. 2012), if viewed as an autonomous sequential decision problem under uncertainty, the problem is not known before the flight. Indeed, the number of targets, zones making up the environment, and positions of targets in these zones, are usually unknown beforehand and must be automatically extracted at the beginning of the mission (for instance using image processing techniques), in order to define the sequential decision problem to optimize. In this paper, we study a target detection and recognition mission by an autonomous UAV, modeled as a POMDP defined during the flight after the number of zones and targets has been online analyzed. We think that this work is challenging and original for at least two reasons: (i) the target detection and recognition mission is viewed as a long-term sequential decision-theoretic planning problem, with both perception actions (changing view angle) and mission actions (moving between zones, landing), for which we automatically construct an optimized policy ; (ii) the POMDP is solved online during the flight, taking into account time constraints required by the mission's duration and possible future execution states of the system.

Achieving such a fully automated mission from end to end requires many technical and theoretical pieces, which can not be all described with highest precision in this paper due to the page limit. We focus attention on the POMDP model, including a detailed discussion about how we statistically learned the observation model from real data, and on the *"optimize-while-execute"* framework that we developed to solve complex POMDP problems online while executing the currently available solution under mission duration constraints. The next section introduces the mathematical model

of POMDPs. In Section 3, we present the POMDP model used for our target detection and recognition mission for an autonomous rotorcraft UAV. Section 4 explains how we optimize and execute the POMDP policy in parallel, dealing with constraints on action durations and probabilistic evolution of the system. Finally, Section 5 presents and discusses many results obtained while experimenting with our approach, showing that Artificial Intelligence techniques can be applied to complex aerial robotics missions, whose decision rules were previously not fully automated nor optimized.

Formal baseline framework: POMDP

A POMDP is a tuple $\langle S, A, \Omega, T, O, R, b_0 \rangle$ where S is a set of states, A is a set of actions, Ω is a set of observations, $T : S \times A \times S \rightarrow [0; 1]$ is a transition function such that $T(s_{t+1}, a, s_t) = p(s_{t+1} \mid a, s_t)$, $O : \Omega \times S \rightarrow [0; 1]$ is an observation function such that $O(o_t, s_t) = p(o_t|s_t)$, $R : S \times A \rightarrow \mathbb{R}$ is a reward function associated with a state-action pair, and b_0 is an initial probability distribution over states. We note Δ the set of probability distributions over the states, called *belief state space*. At each time step t, the agent updates its *belief state* defined as an element $b_t \in \Delta$ using Bayes' rule (Smallwood and Sondik 1973).

Solving POMDPs consists in constructing a policy function $\pi : \Delta \rightarrow A$, which maximizes some criterion generally based on rewards averaged over belief states. In robotics, where symbolic rewarded goals must be achieved, it is usually accepted to optimize the long-term average discounted accumulated rewards from any initial belief state (Cassandra, Kaelbling, and Kurien 1996; Spaan and Vlassis 2004):

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(b_t, \pi(b_t)) \middle| b_0 = b\right] \quad (1)$$

where γ is the actualization factor. The optimal value V^* of an optimal policy π^* is defined by the value function that satisfies the bellman's equation:

$$V^*(b) = \max_{a \in A}\left[\sum_{s \in S} r(s, a)b(s) + \gamma \sum_{o \in O} p(o|a, b)V^*(b_a^o)\right] \quad (2)$$

Following from optimality theorems, the optimal value of belief states is piecewise linear and convex (Smallwood and Sondik 1973), i.e, at a step $n < \infty$, the value function can be represented by a set of hyperplanes over Δ, known as α-vectors. An action $a(\alpha_n^i)$ is associated with each α-vector, that defines a region in the belief state space for which this α-vector maximizes V_n. Thus, the value of a belief state can be defined as $V_n(b) = \max_{\alpha_n^i \in V_n} b \cdot \alpha_n^i$. And an optimal policy in this step will be $\pi_n(b) = a(\alpha_n^b)$.

Recent offline solving algorithms, e.g. PBVI (Pineau, Gordon, and Thrun 2003), HSVI2 (Smith and Simmons 2005), SARSOP (Kurniawati, Hsu, and Lee 2008) and symbolic PERSEUS (Poupart 2005), and online algorithms as RTDP-bel (Bonet and Geffner 2009) and AEMS (Ross and Chaib-Draa 2007) approximate the value function with a bounded set of belief states B, where $B \subset \Delta$. These algorithms implement different heuristics to explore the belief state space, and update the value of V, which is represented by a set of α-vectors (except in RTDP-bel), by a backup operator for each $b \in B$ explored or relevant. Therefore, V is reduced and contains a limited number $|B|$ of α-vectors.

Multi-target detection and recognition mission

Mission description

We consider an autonomous Unmanned Aerial Vehicle (UAV) that must detect and recognize some targets under real-world constraints. The mission consists in detecting and identifying a car that has a particular model among several cars in the scene, and land next to this car. Due to the nature of the problem, especially partially observability due to the probabilistic belief about cars' models, it is modeled as a POMDP. The UAV can perform both high-level mission tasks (moving between zones, changing height level, land) and perception actions (change view angle in order to observe the cars). Cars can be in any of many zones in the environment, which are beforehand extracted by image processing (no more than one car per zone).

The total number of states depends on many variables that are all discretized: the number of zones (N_z), the height levels (H), the view angles (N_Φ), the number of targets ($N_{targets}$) and car models (N_{models}), and a terminal state that characterizes the end of the mission. As cars (candidate targets) can be in any of the zones and be of any possible models a priori, the total number of states is:

$$|S| = N_z \cdot H \cdot N_\Phi \cdot (N_z \cdot N_{models})^{N_{targets}} + T_s$$

where T_s represents the terminal states.

For this application case, we consider 4 possible observations, i.e. $|\Omega| = 4$, in each state: {*car not detected, car detected but not identified, car identified as target, car identified as non-target*}. These observations rely on the result of image processing (described later).

As mentioned before, the high level mission tasks performed by the autonomous UAV are: moving between zones, changing height level, land. The number of actions for moving between zones depends on the number of zones considered. These actions are called $go\_to(\hat{z})$, where $\hat{z}$ represents the zone to go to. Changing the height level also depends on the number of different levels at which the autonomous UAV can fly. These actions are called $go\_to(\hat{h})$, where $\hat{h}$ represents the desired height level. The land action can be performed by the autonomous UAV at any moment and in any zone. Moreover, the land action finishes the mission. We consider only one high level perception action, called *change_view*: change view angle when observing a given car, with two view angles $\Phi = \{front, side\}$. So, the total number of actions can be computed as: $|A| = N_z + H + (N_\Phi - 1) + 1$.

Model dynamics

We now describe the transition and reward models. The effects of each action will be formalized with mathematical equations, which rely on some variables and functions described below, that help to understand the evolution of the POMDP state.

State variables The world state is described by 7 discrete state variables. We assume that we have some basic prior knowledge about the environment:. there are two targets that can be each of only two possible models, i.e. $N_{models} = \{target, non-target\}$. The state variables are:

1. z with N_z possible values, which indicates the UAV's position;
2. h with H possible values, which indicates its height levels;
3. $\Phi = \{front, side\}$, which indicates the view angle between the UAV and the observed car;
4. Id_{target_1} (resp. Id_{target_2}) with N_{models} possible values, which indicates the identity (car model) of target 1 (resp. target 2);
5. z_{target_1} (resp. z_{target_2}) with N_z possible values, which indicates the position of target 1 (resp. target 2).

Transition and reward functions To define the model dynamics, let us characterize each action with:

- *effects*: textual description explaining how state variables change after the action is applied;
- transition function T;
- reward function R.

Concerning the notation used, the primed variables represent the successor state variables, and the variable not primed represent the current state. In addition, let us define the indicative function : $\mathbb{I}_{\{cond\}}$ equal to 1 if condition $cond$ holds, or to 0 otherwise; this notation is used to express the Bayesian dependencies between state variables. Another useful notation is $\delta_x(x')$ equal to 1 if $x = x'$, or to 0 otherwise; this notation allows us to express the possible different values taken by the successor state variable x'.

Based on previous missions with our UAV, we know that moving and landing actions are sufficiently precise to be considered deterministic: the effect of going to another zone, or changing flight altitude, or landing, is always deterministic. However, the problem is still a POMDP, because observations of cars' models is probabilistic ; moreover, it has been proved that the complexity of solving POMDPs essentially comes from probabilistic observations rather than from probabilistic action effects (Sabbadin, Lang, and Ravoanjanahary 2007).

Moreover, in order to be compliant with the POMDP model, which assumes that observations are available after each action is executed, all actions of our model provide an observation of cars' models. The only possible observation after the landing action is *non detected*, since this action does not allow the UAV to take images of the environment. All other actions described in the next automatically take images of the scene available in front of the UAV, giving rise to image processing and classification of observation symbols (see later). As the camera is fixed, it is important to control the orientation of the UAV in order to observe different portions of the environment.

action go_to($\hat{\mathbf{z}}$) This action brings the UAV to the desired zone. The dynamics is described next, but note that if the UAV is in the terminal state (T_s), this action has no effects and no cost (what is not formalized bellow).

- Effects: the UAV moves between zones.
- Transition function:

$$T(s', \texttt{go\_to}(\hat{\texttt{z}}), s) = \delta_{\hat{z}}(z')\delta_h(h')\delta_\Phi(\Phi') \\ \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2})$$

  which, according to the definition of function δ previously mentioned, is non-zero only for the transition where post-action state variables s' are all equal to pre-action state variables s, but the target zone z' that is equal to $\hat{z}$.

- Reward function: $R(s, \texttt{go\_to}(\hat{\texttt{z}})) = C_{z,\hat{z}}$, where $C_{z,\hat{z}} < 0$ represents the cost of moving from z to $\hat{z}$. For this moment we chose to use a constant cost C_z, because actual fuel consumption is difficult to measure with sufficient precision on our UAV. And also, because the automatic generation of the POMDP model does not take into account zone coordinates. Zone coordinates are needed for computing the distance between zones in order to model costs proportionaly to zones' distances.

action go_to($\hat{\mathbf{h}}$) This action leads the UAV to the desired height level. Like action go_to($\hat{\texttt{z}}$), if the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV changes to height level $\hat{\texttt{h}}$.
- Transition function:

$$T(s', \texttt{go\_to}(\hat{\texttt{h}}), s) = \delta_z(z')\delta_{\hat{h}}(h')\delta_\Phi(\Phi') \\ \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2})$$

- Reward function: $R(s, \texttt{go\_to}(\hat{\texttt{h}})) = C_{h,\hat{h}}$, where $C_{h,\hat{h}} < 0$ represents the cost of changing from height level h to $\hat{h}$. This cost also models the fuel consumption depending on the distance between altitudes. These costs are typically higher than costs for moving between zones. For the same reason as the previous action, we also chose to use a constant cost such that $C_z < C_h$.

action change_view This action changes the view angle of the UAV when observing cars. Due to environmental constraints, we assume that all cars have the same orientations in all zones (as in parking lots for instance), so that each view angle value has the same orientation for all zones. Like the previous actions, if the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV switches its view angle (*front* to *side* and vice versa).

- Transition function:

$$T(s', \texttt{change\_view}, s) = \delta_z(z')\delta_{\hat{h}}(h') \\ (\mathbb{I}_{\{\Phi=front\}}\delta_{side}(\Phi') + \mathbb{I}_{\{\Phi=side\}}\delta_{front}(\Phi')) \\ \delta_{Id_{target_1}}(Id'_{target_1})\delta_{z_{target_1}}(z'_{target_1}) \\ \delta_{Id_{target_2}}(Id'_{target_2})\delta_{z_{target_2}}(z'_{target_2})$$

- Reward function: $R(s, \texttt{change\_view}) = C_v$, where $C_v < 0$ represents the cost of changing the view angle. It is represented by a constant cost that is higher than costs of all other actions. Following our previous constant cost assumptions: $C_v \geq C_h > C_z$.

action land This action finalizes the UAV mission, leading the autonomous UAV to the terminal state. If the UAV is in the terminal state (T_s), this action has no effects and no cost.

- Effects: the UAV finishes the mission, and goes to the terminal state.
- Transition function: $T(s', land, s) = \delta_{T_s}(s')$
- Reward function:

$$R(s, \texttt{land}) = \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=target)\}}R_l + \\ \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=target)\}}R_l + \\ \mathbb{I}_{\{(z=z_{target_1})\&(Id_{target_1}=non-target)\}}C_l + \\ \mathbb{I}_{\{(z=z_{target_2})\&(Id_{target_2}=non-target)\}}C_l + \\ \mathbb{I}_{\{(z!=z_{target_1})\&(z!=z_{target_2})\}}C_l$$

where $R_l > 0$ represents the reward associated with a correctly achieved mission (the UAV is in the zone where the correct target is located) and $C_l < 0$ represents the cost of a failed mission. Note that: $R_l \gg C_v \geq C_h > C_z \ggg C_l$.

Observation model

POMDP models require a proper probabilistic description of actions' effects and observations, which is difficult to obtain in practice for real complex applications. For our target detection and recognition missions, we automatically learned from real data the observation model, which relies on image processing. We recall that we consider 4 possible observations in each state: {*no car detected, car detected but not identified, car identified as target, car identified as non-target*}. The key issue is to assign a prior probability on the possible semantic outputs of image processing given a particular scene.

Car observation is based on an object recognition algorithm based on image processing (Saux and Sanfourche 2011), already embedded on-board in our autonomous UAV. It takes as input one shot image (see Fig. 1(a)) that comes from the UAV onboard camera. First, the image is filtered (Fig. 1(b)) to automatically detect if the target is in the image (Fig. 1(c)). If no target is detected, it directly returns the label *no detected*. If a target is detected, the algorithm takes the region of interest of the image (bounding rectangle on Fig. 1(c)), then generates a local projection and compares it with the 3D template silhouettes on a data base of car models (Fig. 1(d)). The local projection only depends on the UAV height level, and camera focal length and azimuth as viewing-condition parameters. The height level is known at every time step, and the focal length and the camera azimuth are fixed parameters. Finally, the image processing algorithm chooses the 3D template that maximizes the similarity (for more details see (Saux and Sanfourche 2011)), and returns the label that corresponds or not to the searched target: *car identified as target* or *car identified as non-target*. If the level of similarity is less than a hand-tuned threshold, the image processing algorithm returns the label *car detected but not identified*.

o_i	$p(o_i\|s)$
car not detected	0.045351
car detected but not identified	0.090703
car identified as target	0.723356
car identified as non-target	0.140590

Table 1: Probability observation table learned from statistical analysis of the image processing algorithm answers using real data, with $s = \{z = z_{target_1}, Id_{target_1} = target, h = 30, z_{target_2} \neq z, Id_{target_2} = non-target\}$.

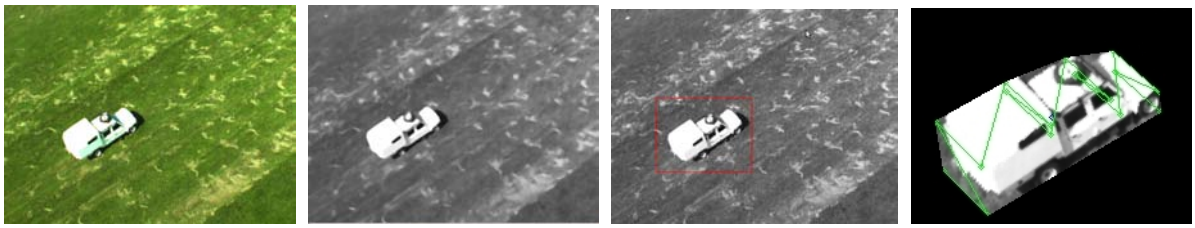
In order to learn the POMDP observation model from real data, we performed many outdoor test campaigns with our UAV and some known cars. It led to an observation model learned via a statistical analysis of the image processing algorithm's answers based on the images taken during these tests. More precisely, to approximate the observation function $O(o_t, s_t)$, we count the number of times that one of the four observations (labels) was an output answer of the image processing algorithm in a given state s. So, we compute $O(o_i, s) = p(o_i|s)$, where o_i is one of the 4 possible observations:

$$p(o_i|s) \simeq \frac{1}{N_{exp}} \sum_{n=1}^{N_{exp}} \mathbb{I}_{\{o_n=o_i|s\}},\ N_{exp} \gg 1.$$

where N_{exp} represents the number of experiences, i.e. the number of runs performed by the image processing algorithm with respect to the different images, and o_n the label obtained at experience n. Note that we have access to more than 500 images for each state, so that $N_{exp} \gg 1$ and the statistical approximations may be good enough. Table 1 shows an example of observation probability obtained after learning in a given state.

Optimize-while-execute framework

Large and complex POMDP problems can rarely be optimized off-line, because of lack of sufficient computational means. Moreover, the problem to solve is not always known in advance, e.g. our target detection and recognition missions where the POMDP problem is based on zones that are automatically extracted from on-line images of the environment. Such applications require an efficient on-line framework for solving POMDPs and executing policies before the mission's deadline. We worked on extending the optimize-while-execute framework proposed



(a) Input image (b) Filtering (c) Car detection (d) Matching

Figure 1: Target detection and recognition image processing based on (Saux and Sanfourche 2011).

in (Teichteil-Konigsbuch, Lesire, and Infantes 2011), previously restricted to deterministic or MDP planning, to on-line solve large POMDPs under time constraints. Our extension is a meta planner that relies on standard POMDP planners like PBVI, HSVI, PERSEUS, AEMS, etc., which are called from possible future execution states while executing the current optimized action in the current execution state, in anticipation of the probabilistic evolution of the system and its environment. One of the issues of our extension was to adapt the mechanisms of (Teichteil-Konigsbuch, Lesire, and Infantes 2011) based on completely observable states, to belief states and point-based paradigms used by many state-of-the-art POMDP planners (Pineau, Gordon, and Thrun 2003; Ross and Chaib-Draa 2007). This framework is different from real-time algorithms like RTDP-bel (Bonet and Geffner 2009) that solve the POMDP *only* from the current execution state, but not from future possible ones as we propose.

We implemented this meta planner with the anytime POMDP algorithms PBVI (Pineau, Gordon, and Thrun 2003) and AEMS (Ross and Chaib-Draa 2007). AEMS is particularly useful for our optimize-while-execute framework with time constraints, since we can explicitly control the time spent by AEMS to optimize an action in a given belief state. The meta planner handles planning and execution requests in parallel, as shown in Fig. 2. At a glance, it works as described in the following:

1. Initially, the meta-planner plans for an initial belief state b using PBVI or AEMS during a certain amount of time (bootstrap).
2. Then, the meta-planner receives an action request, to which it returns back the action optimized by PBVI or AEMS for b.
3. The approximated execution time of the returned action is estimated, for instance 8 seconds, so that the meta planner will plan from some next possible belief states using PBVI or AEMS during a portion of this time (e.g. 2 seconds each for 4 possible future belief states), while executing the returned action.
4. After the current action is executed, an observation is received and the belief state is updated to a new b', for which the current optimized action is sent by the meta-planner to the execution engine.

This framework proposes a continuous planning algorithm that fully takes care of probabilistic uncertainties: it constructs various policy chunks at different future probabilistic execution states.

Furthermore, as illustrated in Fig. 2, planning requests and action requests are the core information exchanged between the main component and the planning component. Interestingly, each component works on an independent thread. More precisely, the main component, which is in charge of policy execution, runs in the execution thread that interacts with the system's execution engine. It competes with the planning component, which is in charge of policy optimization. The planning component runs in the optimization thread that drives the sub-POMDP planner.

Hence, due to thread concurrency, some data must be protected against concurrent memory access with mutexes: planning requests, and the optimized policy. Depending on the actual data structures used by the sub-POMDP planner, read and write access to the policy may be expensive. Therefore, in order to reduce CPU time required by mutex protection and to improve the execution thread's reactivity, we backup the policy after each planning request is solved.

In addition, in real critical applications, end-users often want the autonomous system to provide some basic guarantees. For instance, in case of UAVs, operators require that the executed policy never puts the UAV in danger, what may happen in many situations like being out of fuel. Another danger may come from the lack of optimized action in the current system state, due to the on-line optimization process that has not yet computed a feasible action in this state. For that reason it is mandatory that the meta-planner provides a relevant applicable action to execute when queried by the system's execution scheme according to the current execution state. It can be handled by means of an application-dependent *default policy*, which can be generated before optimization in two different ways: either a parametric off-line expert policy whose parameters are on-line adapted to

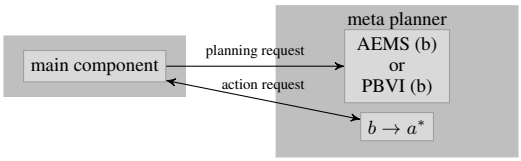


Figure 2: Meta planner planning / execution schema.

the actual problem; or a heuristic policy quickly computed on-line before computing the optimal policy. Simple but complete heuristic POMDP policies, for instance based on the QMDP approximation proposed by (Littman, Cassandra, and Pack Kaelbling 1995), can be quickly generated.

Experimental results

Up to now, we performed complete realistic "hardware in the loop" simulations, i.e. using the exact functional architecture and algorithms used on-board our UAV, a Yamaha Rmax adapted to autonomous flights, as well as real outdoor images. Real flights are being tested at the time we write this article. In this section, we present a deep analysis of results obtained during our realistic simulations.

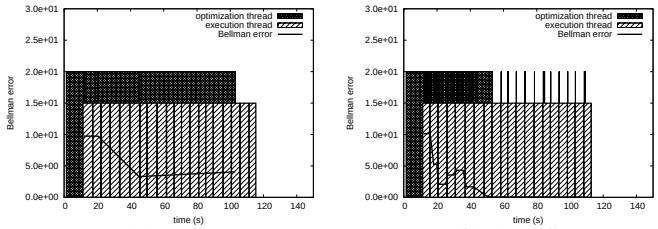
The instance of the problem considered has 2 height levels (30 and 40 meters), 2 view angles (front and side), 2 targets and 2 car models, and 3 zones, which leads to 433 states. Recall that we have 4 observation variables. The aim is to land next to the car whose model is presented in Fig. 1(d); however, the models of the cars is unknown at the beginning of the mission. The meta-planner on-line framework presented in the previous section is a good option for this problem because: (1) the number of zones is discovered in flight, making it impossible to solve the problem before the mission starts, and (2) the POMDP algorithms used – PBVI or AEMS – do not converge within the mission duration limit.

Note that PBVI and AEMS are point-based algorithms that approximate the value function for a set of relevant belief states. PBVI chooses the set of belief states by performing stochastic trials from the initial belief state. AEMS constructs a belief state tree beginning from the initial belief state and expanding this belief tree according to heuristic guidance means.

We consider two initial belief states that represent 2 different initial view angles and the fact that we do not know about the positions and the models of cars: b_0^1 (resp. b_0^2) is a uniform probability distribution over the 12 states $\{z = 1, h = 40, \phi = front, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$ (resp. $\{z = 1, h = 40, \phi = side, z_{target_1} \neq z_{target_2}, Id_{target_1} \neq Id_{target_2}\}$), The reward function is based on the following constants: $C_z = -5$, $C_h = -1$, $C_v = -1$, $R_l = 10$, and $C_l = -100$. The duration of an action is represented by a uniform distribution over $[T^a_{min}, T^a_{max}]$, with $T^a_{min} = 4s$ and $T^a_{max} = 6s$, which is representative of durations observed during preliminary test flights. We recall that we consider static targets.

The observations are characterized by the output of the image processing algorithm (Saux and Sanfourche 2011), which runs in parallel in a concurrent thread, and which is launched as soon as an action is performed. The simulator, which knows the real state of the world, takes an image from the data base and sends it to the image processing algorithm, which returns an observation to the execution component.

Figure 3 shows the timelines for the meta-planner execution process. It represents the periods of time where the policy is optimized (optimization thread) or executed (execution thread) – both running in parallel –, as well as the evolution of the Bellman error during the mission. After a first bootstrap duration (where only the optimization thread is active), we can notice that the optimization process continues for a short time period. Then, small optimization chunks are still processed when new planning requests are sent to the planner, because the policy was previously not fully optimized in the current belief state during previous optimization chunks. The evolution of the Bellman error, reported for each planning request during optimization, emphasizes the evolution of the optimization process. In Fig. 3(a) the value function does not converge for all belief states in the relevant belief set, contrary to 3(b) where the optimization process has converged for the current (sliding) belief state. The reason is that AEMS is more efficient than PBVI, so that it has enough time to optimize the future possible belief states while executing actions. We can notice that the execution thread still goes on, but optimization chunks are very short because the Bellman error is already very small when beginning to optimize from each current belief state.

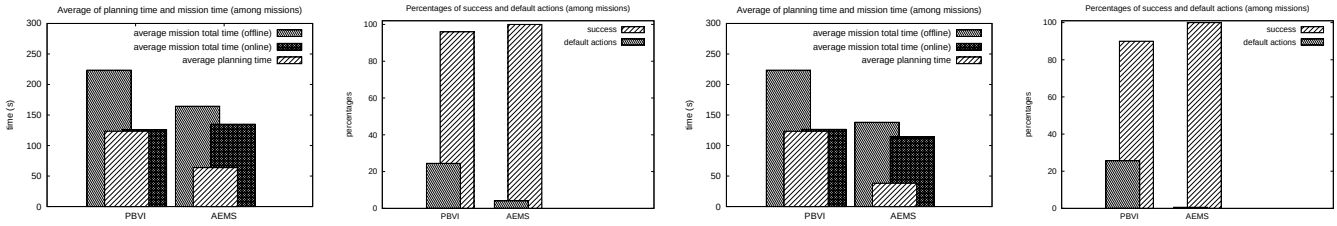


(a) PBVI (b) AEMS

Figure 3: Timelines for PBVI and AEMS implementations of the optimize-while-execute framework starting from b_0^1.

Figure 4 shows results for planning times and mission success percentages, for the 2 underlying POMDP solvers PBVI and AEMS driven by the optimize-while-execute framework: the average mission total time (on-line) represents the time until the end of the mission (i.e. limit time step); the average planning time represents the time taken by the optimization thread, that is very close to the mission total time for the PBVI algorithm, because it cannot converges during the mission time. These average results were computed over 50 test runs for each instance of the problem with a limit horizon of 20 steps ; each test run was a complete mission (optimization and execution in parallel from scratch). To make a comparison, we drown an *offline* mission time that would correspond to optimizing the problem off line (still during the flight after the zones are extracted from the environment in-flight), then executing the optimized policy.

Figure 4 also presents the percentage of default actions and achieved goals. We aim at showing that, depending on the underlying algorithm used (PBVI or AEMS), the planning thread does not react as fast as expected, and more default actions can be performed. We recall that default policy used guarantee reactivity in case the optimized policy is not available in the current execution state. The default policy was quickly computed before computing the optimal policy. We chose a heuristic policy based on the QMDP approximation proposed by (Littman, Cassandra, and Pack Kaelbling 1995).



(a) Planning time for b_0^1 (b) Percentages for b_0^1 (c) Planning time for b_0^2 (d) Percentages for b_0^2

Figure 4: Averaged results for PBVI and AEMS implementations of the optimize-while-execute framework, starting either from belief state b_0^1 or from b_0^2.

The average number of achieved goals (the UAV has landed in the zone containing the car that has the correct target model) is close to 100%, what shows that our approach allows the UAV to achieves its mission very well on average. But, for this kind of partial observable problem where the real nature of the targets is not known, and where the observation model is not exactly due to imprecision of the observation model learned from the image processing algorithm, we think that if targets positions are not static it may be impossible to achieve the goal 100% of the time.

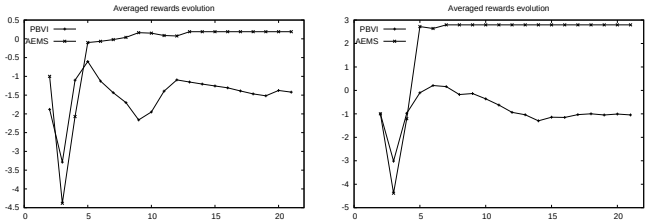
Figures 5(a) and 5(b) present the averaged return taken over 50 real policy executions, statistically computed as:

$$V^{\pi}(s_t) = \frac{1}{50} \sum_{50} \left[\sum_{k=0}^{t} \gamma^k r(s_k, \pi(b_k)) | b_0, s_k \right] \qquad (3)$$

Note that the simulator uses its knowledge about the environent (i.e. the state s_t and all s_k), to attibuate the rewards while simulating. This equation allows us to show the accumulated rewards from the time step zero until time step t.

For PBVI, regardless of the initial belief state, the average return gathered during policy execution tends to be less important than for AEMS. We believe that this difference comes from the fact that PBVI is less reactive (efficient) than AEMS so that more default actions are performed, which are not optimal for the belief in which they were applied.

Finally, we counted the number of times that a $change\_view$ action was chosen by the policy, in order to evaluate the impact of the "perception action". For the initial belief state b_0^1 (i.e. $\Phi = front$), this action was chosen 2 times over 50 runs for PBVI, and also 2 times for AEMS. And, for b_0^2 (i.e. $\Phi = side$): 50 times with PBVI and 50 times for AEMS too. We believe that this behavior comes from the observation model, which is more discriminative when $\Phi = front$ than when $\Phi = side$: from an initial belief with $\Phi = front$, the policy optimization algorithm does not find interesting to change the observation view angle.



(a) Averaged rewards for b_0^1 (b) Averaged rewards for b_0^2

Figure 5: Average rewards for PBVI and AEMS implementations of the optimize-while-execute framework, starting either from belief state b_0^1 or from b_0^2.

Conclusion and future work

To the best of our knowledge, this paper presents one of the first *POMDP-based* implementations of a target detection and recognition mission by an autonomous rotorcraft UAV. Our contribution is threefold: (i) we model perception and mission actions in the same decision formalism using a single POMDP model; (ii) we statistically learn a meaningful probabilistic observation model of the outputs of an image processing algorithm that feeds the POMDP model; (iii) we provide practical algorithmic means to optimize and execute POMDP policies in parallel under time constraints, what is required because the POMDP problem is generated during the flight. We analyzed experiments conducted with a realistic "hardware in the loop" simulation based on real data, which demonstrate that POMDP planning techniques are now mature enough to tackle complex aerial robotics missions, on condition of using some kind of "optimize-while-execute" framework, as the one proposed in this paper.

At the time of writing this paper, we are embedding our decision-making components on-board the real UAV and beginning to conduct real outdoor flights. Many improvements can be considered for future research: analyzing the impact of different initial belief states on the optimized strategy; taking into account safety constraints imposed by civil aeronautical agencies when optimizing the strategy; building POMDP policies that are robust to imprecise observation models.

References

Bai, H.; Hsu, D.; Kochenderfer, M.; and Lee, W. S. 2011. Unmanned Aircraft Collision Avoidance using Continuous-State POMDPs. In *Proceedings of Robotics: Science and Systems*.

Bonet, B., and Geffner, H. 2009. Solving POMDPs: RTDP-bel vs. point-based algorithms. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, 1641–1646. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Candido, S., and Hutchinson, S. 2011. Minimum uncertainty robot navigation using information-guided POMDP planning. In *ICRA'11*, 6102–6108.

Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ*.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. RSS*.

Littman, M.; Cassandra, A.; and Pack Kaelbling, L. 1995. Learning policies for partially observable environments: Scaling up. In *International Conference on Machine Learning*, 362–370.

Miller, S. A.; Harris, Z. A.; and Chong, E. K. P. 2009. A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP J. Adv. Signal Process* 2009:2:1–2:17.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. of IJCAI*.

Poupart, P. 2005. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. Ph.D. Dissertation, University of Toronto.

Ross, S., and Chaib-Draa, B. 2007. AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2592–2598.

Sabbadin, R.; Lang, J.; and Ravoanjanahary, N. 2007. Purely epistemic markov decision processes. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, 1057–1062. AAAI Press.

Saux, B., and Sanfourche, M. 2011. Robust vehicle categorization from aerial images by 3d-template matching and multiple classifier system. In *7th International Symposium on Image and Signal Processing and Analysis (ISPA)*, 466–470.

Schesvold, D.; Tang, J.; Ahmed, B.; Altenburg, K.; and Nygard, K. 2003. POMDP planning for high level UAV decisions: Search vs. strike. In *In Proceedings of the 16th International Conference on Computer Applications in Industry and Enineering*.

Smallwood, R., and Sondik, E. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 1071–1088.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. UAI*.

Spaan, M., and Vlassis, N. 2004. A point-based POMDP algorithm for robot planning. In *ICRA*.

Spaan, M. 2008. Cooperative Active Perception using POMDPs. *Association for the Advancement of Artificial Intelligence - AAAI*.

Teichteil-Konigsbuch, F.; Lesire, C.; and Infantes, G. 2011. A generic framework for anytime execution-driven planning in robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 299–304.

Wang, J.; Zhang, Y.; Lu, J.; and Xu, W. 2012. A Framework for Moving Target Detection, Recognition and Tracking in UAV Videos. In Luo, J., ed., *Affective Computing and Intelligent Interaction*, volume 137 of *Advances in Intelligent and Soft Computing*. Springer Berlin / Heidelberg. 69–76.

On Estimating the Return of Resource Aquisitions through Scheduling: An Evaluation of Continuous-Time MILP Models to Approach the Development of Offshore Oil Wells

Thiago Serra and **Gilberto Nishioka** and **Fernando J. M. Marcellino**
PETROBRAS – Petróleo Brasileiro S.A., Av. Paulista, 901, 01311-100, São Paulo – SP, Brazil
{thiago.serra, nishioka, fmarcellino}@petrobras.com.br

Abstract

Resources such as oil rigs and pipelay vessels represent the bottleneck in the development of offshore oil wells. Being expensive to hire, oil companies expect a short-term payback from their use. Consequently, the immediate production brought by their scheduling is intended to be maximized in the Offshore Resources Scheduling Problem (ORSP), and the comparison of optimal solutions to instances differing by their availability can be used to aid the decision-making process for hiring more resources. Regarding that it is often difficult to find an optimal solution, we attempt to use estimations obtained by solving a relaxation instead. This paper describes the evaluation of alternatives to model the ORSP without inventory constraints, and discusses how to eventually include them. Those alternatives are continuous-time MILP models in which decisions related to the sequencing of activities are not indexed by resources. They differ from each other by the constraints to avoid the occurrence of simultaneous activities and by the addition of cuts. According to the tests performed, tighter upper bounds to instances from the literature were achieved either by finding an optimal solution to such relaxation or by the upper limit reached at the solver halt.

Introduction

The development of an offshore oil well is comprised of a sequence of activities with varied roles, each of which requiring distinct capabilities from the resource that performs it. Those activities represent consecutive stages in the development of each well, such as its drilling, completion, and connection to a producing unit. The availability of certain resources represents the development bottleneck due to their scarcity in face of the number of activities simultaneously demanding their use. Among the most required and expensive resources there are oil rigs and pipelay vessels. The former type of resource is required by the early stages of drilling and completion, while the latter performs the connection of wells to producing units through pipes previously loaded at harbors. The criterion of main concern to the schedule of offshore resources is the maximization of the short-term production, which is measured by how much each well produces from the day its development is concluded until a given planning horizon. Such criterion enables the comparison of different solutions by measuring how each of them impacts the immediate oil production. Besides, solutions to instances corresponding with scenarios that differ only by the availability of resources can be compared to evaluate how acquiring or leasing additional resources impact the production and thus justifies, or not, the investment required. While it is not always possible to achieve an optimal solution in reasonable time to such kind of problem, such analysis can be quite inaccurate and misleading if the schedules that are being compared are not optimal. In such a case, the relaxation of some constraints allows to solve a simplified version of the problem to optimality, and therefore set an upper bound to the achievable production.

The primary aim of this work is to illustrate the importance of striving for accurate assessments of return on resources by means of scheduling models. It will be done by comparing the results of the approach developed here with production bounds previously reported to the Offshore Resources Scheduling Problem (ORSP). Consequently, we also expect to achieve tighter upper bounds to the ORSP while exploring Mixed-Integer Linear Programming (MILP) models aimed at solving its relaxation. The analysis and comparison of modeling alternatives is intended to eventually direct the extension of one of such models to tackle the full problem, and thus attempt to solve it to optimality.

The ORSP has already been tackled by many approaches since its inception by Hasle et al. (1996). Several of them (do Nascimento 2002; Accioly, Marcellino, and Kobayashi 2002; Pereira, Moura, and de Souza 2005; Moura, Pereira, and de Souza 2008; Serra, Nishioka, and Marcellino 2011) represent a progressive detail of a single specification, which is related to a software developed and used by the Brazilian oil company Petrobras. do Nascimento showed that the ORSP belongs to the NP-Hard class for being as hard as the Job-shop Scheduling Problem (JSP). Thus, an efficient algorithm for the general case of the problem is not known. On account of the size of the instances for which a solution was being pursued, techniques with smaller memory requirements but focused on feasibility instead of optimality such as Constraint Programming (CP) and metaheuristics have been applied to some extent. Nevertheless, an algorithm to generate an upper bound was introduced by Serra,

Nishioka, and Marcellino and it proved that their CP approach was able to find solutions at most 40% far from the optimal in all tested instances. However, they acknowledged that the estimation provided was very conservative and probably not accurate enough to evaluate the solutions obtained. In this work, we intend to tighten such estimation by approaching the ORSP without loading activities. Those activities are required to load pipes at harbors prior to the connection activities in which they are released by vessels. They were included in the ORSP by Serra, Nishioka, and Marcellino (2011). Hence, our approach targets the problem as it was described until 2008 by Moura, Pereira, and de Souza.

The selection of MILP to tackle the ORSP relaxation derives from the fact that our focus was on approaching the optimal production instead of a feasible schedule. Following the trend observed by Floudas and Lin (2004) in the development of scheduling models, the temporal decisions were represented using continuous variables instead of binary variables indexed by discrete time units. That led to a short representation of precedence relations between activities and to a model which, in absence of resource and positioning-related constraints, requires a single binary variable per activity regardless of the size of the scheduling makespan. Although the number of variables is usually smaller in continuous-time models, Stefansson et al. (2011) noticed that the number of constraints tends to be larger and that they are more complex due to the difficulty of representing certain resource restrictions without time indexing. Hence, one of the main concerns of developing such kind of model is establishing a set of constraints binding resource assignment to activities sequencing that facilitates the resolution while keeping the models as small as possible. In order to address the issue of the model size, there were developed at least two types of models in which variables for sequencing activities are not indexed by resources. Each of them depicts the same problem with a different meaning for the decision variables required by the constraints that avoid the occurrence of simultaneous activities on the resources. The first type represents the sequencing decisions detached from the precise order of execution, and thus allow one activity to be preceded by multiple activities and vice versa (e.g., Jain and Grossmann (2001)). The other type considers sequencing solely as the immediate succession of activities on each resource (e.g., Méndez, Henning, and Cerdá (2000), and Gupta and Karimi (2003)). Since it is not possible to tell in advance which type is more suitable as well as whether the addition of specific cuts in advance like reported by Jain and Grossmann (2001) is effective to assist the solving process in cases like ours, we will proceed to an evaluation of some modeling alternatives to that part of the model.

The organization of the remainder of the paper is as follows. The next section formally defines the ORSP relaxation that will be approached. We then present a MILP model covering part of the details and proceed to the description of some alternatives to model the remainder. Results from tests involving the models are shown afterwards. They are followed by a brief discussion and a conclusion regarding what was achieved, as well as how to improve and extend the present approach to tackle the full problem.

Problem Definition

The description that follows defines the relaxation of the Offshore Resources Scheduling Problem (ORSP) that we aim to solve. We consider only well development activities that must be performed at well sites by rigs and vessels. Each of those activities must be assigned to a resource compatible with its needs, thus comprising requirements such as being able to operate at the depth of the well. Rigs and vessels are required to attend to a large geographical area, in which displacements between sites may take a considerable time.

The ORSP can be depicted as a matter of deciding *if*, *when* and *how* to perform each of several activities using a set of resources in a short-term period. Time is represented in days, starting from a date set as 0. For notational convention, resources will be denoted by index **i** assuming values in the set **I**, and activities by index **j** assuming values in the set **J**.

Optimization criteria

We want to maximize the short-term production of the developments, which is a measure of how much each well would produce since the day its development finishes until the a given horizon date **H**. Thus, we consider that each activity j induces a daily production rate $\mathbf{pr_j}$ once it is concluded, which is nonzero only for the last activity of each well.

Resource constraints

- Resources like rigs and vessels are unary, meaning that each of them performs at most one activity at a time.
- A resource must be assigned to each activity that will be concluded before the horizon limit.
- A resource i can be assigned to perform an activity j if, and only if, $\mathbf{c_{ij}} = 1$.
- Each resource i has a contractual period of use, ranging from its release date $\mathbf{rr_i} \geq 0$ to its deadline $\mathbf{rd_i} \geq rr_i$.
- During that contract period, there are predicted periods of unavailability. Without loss of generality, they can be mapped as activities.
- Only one resource can be assigned to perform an activity on a well at any time.
- To perform consecutive activities j_1 and j_2 on distinct wells, it must be accounted the displacement time $\mathbf{dt_{j_1 j_2}}$.

Activity constraints

- Well development activities are non-preemptive: they are performed without interruption until their conclusion.
- Each activity j has to be scheduled between its release date $\mathbf{ar_j} \geq 0$ and its deadline $\mathbf{ad_j} \geq ar_j$.
- Each activity j is associated with a well denoted by $\mathbf{w_j}$.
- Each activity j requires $\mathbf{p_j}$ days to be processed.
- One activity may be preceded by other activities. Let $\mathbf{pc_{j_1 j_2}} = 1$ if, and only if, activity j_2 is preceded by j_1 and $\mathbf{pd_{j_1 j_2}}$ be the minimum delay in days between both.
- Some activities may belong to a cluster, in which all activities should be performed by a single resource. Let $\mathbf{cl_j}$ denote the index of the cluster of each activity j.

Model Description

There is a common representation for time variables, precedence relations and part of the resource constraints for all models considered. The set of decision variables involved is described in table 1. In the following, the objective function and the constraints of such common part are stated.

Var.	Domain	Description
S_j	$\mathbb{R}^+$	Start time of activity j.
E_j	$\mathbb{R}^+$	End time of activity j.
Y_j	$\{0,1\}$	If activity j ends before H.
X_j	$\mathbb{R}^+$	Days before j concludes if $Y_j = 1$.
F_j	$\mathbb{R}^+$	Auxiliary variable positive if $Y_j = 0$.
W_{ij}	$\{0,1\}$	If resource i performs activity j.
$Z_{j_1 j_2}$	$\{0,1\}$	If activity j_1 precedes activity j_2.

Table 1: Decision variables shared by all models considered.

The objective function (1) is aimed to maximize the production until day H.

$$\text{max.} \sum_{j \in J} (H \times Y_j - X_j) \times pr_j \quad (1)$$

While it is possible to represent a negative production with such function if $X_j > 0$ and $Y_j = 0$ for an activity j, that does not occur when the problem is solved because a higher return would be achieved with $X_j = 0$ and $F_j = E_j$. Thus, the partial solution at any branch of the search tree of the problem will always set $X_j = 0$ in such a case.

For constraints (6), (14) and (17) in the following, there is a coefficient M_j associated with each activity j. Such coefficient is intended to be larger than any valid assignment to E_j. This kind of coefficient is usually described in the literature as "Big M". Despite being possible to state $M_j = ad_j$, setting a smaller value would facilitate the solving process. Thus, a preliminary run of the model comprised of constraints (2) to (19) was made for each instance in order to update M_j as $\overline{E_j} + H - 1$, where $\overline{E_j}$ is the value obtained for E_j in such run. The interested reader is referred to the work of Camm, Raturi, and Tsubakitani (1990) for an empirical investigation on the scalability issues incurred by the selection of overly large values for such kind of coefficient.

Constraints (2) to (4) limit the time variables from sets S and E by release date, deadline, and processing time.

$$\text{s.t.} \quad S_j + p_j = E_j, \qquad \forall j \in J \quad (2)$$
$$S_j \geq ar_j, \qquad \forall j \in J \quad (3)$$
$$E_j \leq ad_j, \qquad \forall j \in J \quad (4)$$

The semantics of the variable sets X and Y as described in table 1 is guaranteed by constraints (5) to (7). Thus, $F_j > 0$ if, and only if, $Y_j = 0$ for any activity j. Otherwise, $X_j = E_j$. Therefore, the variable sets X and Y can be used in the objective function to represent the production achieved.

$$E_j = X_j + F_j, \qquad \forall j \in J \quad (5)$$
$$F_j \leq M_j \times (1 - Y_j), \qquad \forall j \in J \quad (6)$$
$$E_j \geq H \times (1 - Y_j), \qquad \forall j \in J \quad (7)$$

The constraint (8) is a cut that was included to facilitate the solving process. It consists of an additional binding between the sets Y and X in order to evidence when an activity cannot be performed before day H.

$$X_j + Y_j \leq H, \qquad \forall j \in J \quad (8)$$

Further information about cuts will be provided in the section discussing the preliminary addition of cuts.

Precedence relations are enforced by equation (9) on the variable sets S and E, as well as by the additional cut (10) on Y. Such cut restrains the occurrence of an activity before H to the case in which its predecessors also occur before H.

$$E_{j_1} + pd_{j_1 j_2} \leq S_{j_2}, \qquad \forall j_1, j_2 \in J, pc_{j_1 j_2} = 1 \quad (9)$$
$$Y_{j_1} \geq Y_{j_2}, \qquad \forall j_1, j_2 \in J, pc_{j_1 j_2} = 1 \quad (10)$$

Equation (11) guarantees that each activity concluded before H is assigned to a resource, and that such resource is compatible with the activity as well.

$$\sum_{i \in I, c_{ij} = 1} W_{ij} = Y_j, \qquad \forall j \in J \quad (11)$$

Equation (12) ensures that the total processing time of the activities assigned to each resource does not exceed its short-term availability.

$$min\{H - 1, rd_i\} - min\{H - 1, rr_i\} \geq \sum_{j \in J, c_{ij} = 1} p_j \times W_{ij}, \qquad \forall i \in I \quad (12)$$

Equations (13) and (14) limit the scheduling of each activity to the contract of the resource it is assigned to.

$$S_j \geq rr_i \times W_{ij}, \qquad \forall i \in I, j \in J, c_{ij} = 1 \quad (13)$$
$$E_j \leq rd_i + M_j \times (1 - W_{ij}), \qquad \forall i \in I, j \in J, c_{ij} = 1 \quad (14)$$

Constraints (15) to (17) restrain the simultaneous occurrence of activities. Up to this point, that only applies to those performed on the same well. The transition time of (17) is null in such a case. Such transition time serves to separate activities from distinct wells that use the same resource.

$$Z_{j_1 j_2} + Z_{j_2 j_1} \leq 1, \qquad \forall\, j_1, j_2 \in J, j_1 < j_2 \quad (15)$$
$$Z_{j_1 j_2} + Z_{j_2 j_1} \geq Y_{j_1}, \qquad \forall\, j_1, j_2 \in J, j_1 \neq j_2, w_{j_1} = w_{j_2} \quad (16)$$
$$E_{j_1} + dt_{j_1 j_2} - (M_{j_1} + dt_{j_1 j_2}) \times (1 - Z_{j_1 j_2}) \leq S_{j_2}, \qquad \forall\, j_1, j_2 \in J, j_1 \neq j_2 \quad (17)$$

The assignment of activities belonging to a cluster is restricted to resources capable of performing all of those scheduled in the short-term by constraint (18) and to a single resource by (19).

$$W_{ij_1} \leq 1 - Y_{j_2}, \qquad \forall i \in I, j_1, j_2 \in J, c_{ij_1} = 1, c_{ij_2} \neq 1, cl_{j_1} = cl_{j_2} \quad (18)$$
$$W_{ij_1} \leq 1 - Y_{j_2} + W_{ij_2}, \qquad \forall\, i \in I, j_1, j_2 \in J, c_{ij_1} = 1, c_{ij_2} = 1, cl_{j_1} = cl_{j_2} \quad (19)$$

Full resource sequencing constraints

Our first alternative to complete the model presented above is by means of a full resource sequencing formulation. In such a case, whenever two activities j_1 and j_2 are performed by a single resource, either $Z_{j_1 j_2} = 1$ or $Z_{j_2 j_1} = 1$ in order to avoid their simultaneous occurrence.

Constraint (20) denotes a possibility to represent such restriction inspired on the model described by Jain and Grossmann (2001). It forces the condition above when two activities are assigned to a single resource.

$$Z_{j_1 j_2} + Z_{j_2 j_1} \geq W_{ij_1} + W_{ij_2} - 1, \quad \forall i \in I, j_1, j_2 \in J, c_{ij_1} = 1, c_{ij_2} = 1 \; j_1 < j_2 \quad (20)$$

The model comprising such constraint along with the others stated previously will be denoted as $\mathbf{M}_F$ hereafter.

Immediate resource sequencing constraints

Another possibility to complete the model of the same problem is to use immediate sequencing instead of full resource sequencing to constrain the concurrence of activities on each resource. In such a case, we must have auxiliary decision variables that explicitly define which activity precedes each other or else if an activity is the first to be performed by a resource. Those sets of decision variables are depicted in table 2. They are named W' and Z' due to their complementary nature with regard to W and Z, respectively. As observed by Méndez, Henning, and Cerdá (2000), their integrality restriction can be relaxed because it is not necessary in such context. The constraints below are based on those used by Méndez, Henning, and Cerdá (2000).

Var.	Domain	Description
W'_{ij}	$\mathbb{R}^+$	If j is the first activity on resource i.
$Z'_{j_1 j_2}$	$\mathbb{R}^+$	If activity j_2 immediately follows j_1.

Table 2: Decision variables for immediate sequencing.

Constraints (21) and (22) limit the values of W' and Z' by W and Z.

$$W'_{ij} \leq W_{ij}, \quad \forall i \in I, j \in J, c_{ij} = 1 \quad (21)$$

$$Z'_{j_1 j_2} \leq Z_{j_1 j_2}, \quad \forall j_1, j_2 \in J, j_1 \neq j_2 \quad (22)$$

Constraint (23) states that at most one activity is the first on each resource, and (24) that each activity is succeeded by at most a single activity.

$$\sum_{j \in J, c_{ij}=1} W'_{ij} \leq 1, \quad \forall i \in I \quad (23)$$

$$\sum_{j_2 \in J, j_2 != j_1, \exists i \in I, c_{ij_1}=1, c_{ij_2}=1} Z'_{j_1 j_2} \leq 1, \quad \forall j_1 \in J \quad (24)$$

Constraint (25) ensures that each activity scheduled before H is either the first on a resource or succeeds another one.

$$\sum_{i \in I, c_{ij_1}=1} W'_{ij_1} + \sum_{j_2 \in J, j_2 != j_1, \exists i \in I, c_{ij_1}=1, c_{ij_2}=1} Z'_{j_2 j_1} = Y_j, \quad \forall j \in J \quad (25)$$

Constraint (26) defines that a transition can only occur if a pair of activities are assigned to the same resource, while (27) avoids a transition if one activity is assigned to a resource incompatible with the other activity.

$$W_{ij_1} \leq W_{ij_2} + 1 - Z'_{j_1 j_2}, \quad \forall i \in I, j_1, j_2 \in J \; c_{ij_1} = 1, c_{ij_2} = 1 \quad (26)$$

$$W_{ij_1} \leq 1 - Z'_{j_1 j_2}, \quad \forall i \in I, j_1, j_2 \in J, c_{ij_1} = 1, c_{ij_2} \neq 1 \quad (27)$$

Along with statements (1) to (19), the constraints depicted in this section constitute model $\mathbf{M}_I$.

Preliminary addition of cuts

Cuts are inferred constraints that can only be obtained from the model constraints by leveraging integrality restrictions. They are not required to ensure the model correctness but they help pruning the problem space towards the polytope of integer solutions, thus facilitating the solving process.

Jain and Grossmann (2001) suggested a cut for unary scheduling that consists of avoiding the sequencing of activities associated with different resources. However, it was prohibitive in our case to consider every combination of activities and resources pairs. Alternatively, constraint (28) is adapted from another proposed by Gupta and Karimi (2003). It forbids the sequencing of a pair of activities if one of them is assigned to a resource that is incompatible with the other.

$$\sum_{i \in I, c_{ij_1}=1, c_{ij_2} \neq 1} W_{ij_1} + \sum_{i \in I, c_{ij_2}=1, c_{ij_1} \neq 1} W_{ij_2} + 2 \times (Z_{j_1 j_2} + Z_{j_2 j_1}) \leq 2, \quad \forall j_1, j_2 \in J, w_{j_1} \neq w_{j_2} \quad (28)$$

It is worth of notice that the space pruning incurred by such constraint is a subset of the pruning that the constraint of Jain and Grossmann could achieve. Based on it, we have developed constraint (29) to provide the same cut suggested by Jain and Grossmann, but using less combinations.

$$(Z_{j_1 j_2} + Z_{j_2 j_1}) + \sum_{i_2 \in I, i_2 \neq i_1, c_{ij_2}=1, c_{ij_1}=1} W_{ij_2} + W_{i_1 j_1} \leq 2, \quad \forall i_1 \in I, j_1, j_2 \in J, c_{i_1 j_1} = 1, c_{i_1 j_2} = 1 \; w_{j_1} \neq w_{j_2} \quad (29)$$

Let $\mathbf{M}_{FC}$ be the model resulting from the addition of (28) to M_F and $\mathbf{M}_{F*}$ the one including (29) instead. Similarly, models $\mathbf{M}_{IC}$ and $\mathbf{M}_{I*}$ are defined by the addition of constraints (28) and (29) to M_I, respectively.

Experimental Evaluation

The models previously described were tested using the set of instances from Serra, Nishioka, and Marcellino (2011). That set represents a past scenario comprising the activities to develop 171 wells using 73 resources.

Table 3 summarizes how many activities (Acs.), wells (Wss.), rigs (Rgs.) and vessels (Vss.) each instance (Ins.) has. Instance O contains the entire set of activities, which is partitioned approximately into halves for instances $H1$ and $H2$ as well as into quarters for instances $Q1$ to $Q4$. The table also shows the former production estimations (F.P.E.) recorded, which represent an upper bound to the achievable production of each instance. For the sake of comparability, those values are normalized considering the production of the best ORSP solution found for each instance on the experiments of Serra, Nishioka, and Marcellino as 100.

Ins.	Q1	Q2	Q3	Q4	H1	H2	O
Acs.	116	118	116	115	231	234	465
Wls.	46	37	45	43	82	89	171
Rgs.	64						
Vss.	9						
F.P.E.	122.08	138.13	117.78	118.31	122.32	124.94	138.24

Table 3: Instances characteristics and former limits.

The models were implemented in the OPL language and run using the IBM Cplex Studio 12.2 solver with some customizations. The time limit was set as one hour in accordance to the end user expectation. The relative precision gap was set as 0.005% to guarantee precision up to the second decimal digit. Based on preliminary tests, the solver was set to emphasize optimality over feasibility and to repeat presolve with cuts and allow new root cuts (IBM 2010). The computer used had 4 Dual-Core AMD Opteron 8220 processors, 16 Gb of RAM and a Linux operating system.

Each instance was run twice for each model with different starting solutions. The first one consists of an empty short-term schedule and the second is the best solution found by Serra, Nishioka, and Marcellino (2011). Despite helping the solving process, those solutions were seldom valid due to the smaller values set for the M coefficients, which in turn reduced the time for scheduling after the horizon.

Tables 4 and 5 show either the time to achieve a solution to the ORSP relaxation with the required gap, the gap assured to the best solution found at the solver halt, or a dash if the run was unsuccessful. Table 6 reports the tightest production estimation (T.P.E.) achieved for each instance as a result of those runs. It corresponds to the optimal solution of the ORSP relaxation that we approached for instances $Q1$ to $Q4$ and to the smaller upper limit recorded for the others.

	Q1	Q2	Q3	Q4	H1	H2	O
M_F	37 s	21 s	35 s	49 s	0.06%	0.15%	30%
M_{FC}	28 s	30 s	201 s	69 s	0.36%	0.42%	111%
M_{F*}	244 s	621 s	289 s	203 s	37%	33%	124%
M_I	504 s	2425 s	1940 s	810 s	–	–	–
M_{IC}	448 s	3147 s	974 s	753 s	–	–	–
M_{I*}	6.6%	18%	1405 s	0.27%	–	–	–

Table 4: Solving time (s) or gap (%) after 1 hour for each model-instance run starting from an empty schedule.

	Q1	Q2	Q3	Q4	H1	H2	O
M_F	12 s	10 s	14 s	11 s	0.08%	0.16%	9.4%
M_{FC}	13 s	12 s	15 s	28 s	0.29%	0.62%	182%
M_{F*}	177 s	256 s	112 s	110 s	1.4%	1.7%	14%
M_I	93 s	450 s	293 s	192 s	–	1.6%	–
M_{IC}	303 s	124 s	91 s	236 s	–	–	–
M_{I*}	0.52%	546 s	567 s	1883 s	186%	193%	–

Table 5: Solving time (s) or gap (%) after 1 hour for each model-instance run starting from a feasible solution.

Ins.	Q1	Q2	Q3	Q4	H1	H2	O
T.P.E.	104.04	113.90	100.62	103.53	105.00	105.91	118.06

Table 6: Tightest production estimation found for each instance with an optimal solution or an upper limit at the halt.

Discussion

From the comparison of tables 3 and 6, we observe a reduction in at least a half of the optimality gap of the best solution found for each instance by Serra, Nishioka, and Marcellino (2011). The production limit estimation for instance Q_3 was reduced from 117.78 to 100.62. Instances H_1 and H_2 also presented considerable improvements with the gap reduction of their best solutions to less than a quarter of the former. However, it is worth noticing that instances of similar size such as Q_1 to Q_4 had reductions of varied proportions. Hence, the variability in the reduction of the estimations endorses the use of more accurate relaxations in order to obtain reliable production assessments.

Regarding the variance of performance between different models, those using full resource sequencing had better results than the others using immediate sequencing. That may be explained by the need of keeping two sets of sequencing variables in the latter type of models due to the constraints to limit the concurrence on each well. Moreover, the preliminary addition of cuts often had a negative effect in the processing effort. Therefore, we conclude that model M_F represents the most promising option among those evaluated to the development of an extended approach to the ORSP.

Conclusion and Future Work

This work aimed at evidencing the strategic importance of scheduling applications by means of a study case. It consisted of tightening the production estimation from using oil rigs and pipelay vessels to develop offshore oil wells. Such analysis can be applied to other industries where resource acquisition figures are in similar orders of magnitude. In all

cases, perceiving scheduling merely as an operational tool hinders a better valuation of machinery, and thus has the potential of leading to bad managerial decisions.

We also aimed at depicting a solving approach based on the discussion of relevant MILP modeling issues such as time representation, sequencing of activities in resources and preliminary addition of cuts. While the time representation is discussed according to the review of Floudas and Lin (2004) and to the evaluation conducted by Stefansson et al. (2011), the latter issues are further scrutinized by the experimental comparison of varied models inspired by the literature. Those models differ from each other in two ways. First, they vary in regard to the selection between full sequencing and immediate sequencing of the activities assigned to each resource. We have observed that the selection of constraints for sequencing in continuous-time MILP models is not much discussed. However, as reported by our experimental evaluation, it has a great impact in the models performance. Second, some of them rely on a preliminary addition of cuts aimed to improve the solving time. Such addition was motivated by the benefit of the practice when approaching a similar problem, as described by Jain and Grossmann (2001). Comparing them, we observed that the modeling of a full resource sequencing without further cuts was the alternative that yielded better results. In addition, it was observed that leveraging good solutions as a starting point to the solver improved considerably the outcome. Despite the fact that the modeling evaluation relied on the straightforward use of a MILP solver, it evidences the current capabilities of such solvers and provides guidance to approaches using MILP scheduling models as building blocks to tackle more complex problems. The results achieved represent improved upper bounds to the ORSP when inventory constraints are also considered. Nonetheless, they also constitute optimal or approximate solutions to the problem as it was described in the literature until 2008, when such constraints were absent. Hence, we discussed scheduling applications and modeling issues that can help practitioners to tackle similar problems.

Future work will be directed towards reducing the time necessary to solve bigger instances as well as extending the current approach in order to generate feasible schedules. The improvement induced by the provision of a starting solution from a CP approach evidences the possible benefit of a hybridization of techniques. On the one hand, developing a CP model fully compliant with the relaxation described in this work would guarantee the acceptance of the initial solution provided to the MILP model and thus reduce the effort to solve the proposed relaxation. On the other hand, the optimal solution to the relaxation could also be used somehow as an input to a technique targeting feasibility and therefore aid with generating valid schedules. Alternatively, the model described could be split into models aimed at tackling different decisions and then integrated by means of a decomposition approach iteratively switching between them. For instance, Hooker (2004) describes a Logic-Based Benders Decomposition (LBBD) in which decisions related to resource assignment are taken by a MILP model and those related to activities sequencing are taken by a CP model. Based on accounts of previous CP approaches, it appears to be a promising line.

Acknowledgments The authors gratefully acknowledge the company under study for authorizing the publication of the information here present. Furthermore, the opinions and concepts presented are the sole responsibility of the authors.

References

Accioly, R.; Marcellino, F. J. M.; and Kobayashi, H. 2002. Uma aplicação da programação por restrições no escalonamento de atividades em poços de petróleo. In *Proceedings of the 34th Brazilian Symposium on Operations Research*.

Camm, J. D.; Raturi, A. S.; and Tsubakitani, S. 1990. Cutting big M down to size. *Interfaces* 20:61–66.

do Nascimento, J. M. 2002. Ferramentas computacionais híbridas para a otimização da produção de petróleo em águas profundas. Master's thesis, Unicamp, Brazil.

Floudas, C. A., and Lin, X. 2004. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. *Computers and Chemical Engineering* 28:2109–2129.

Gupta, S., and Karimi, I. A. 2003. An improved MILP formulation for scheduling multiproduct, multistage batch plants. *Industrial & Engineering Chemistry Research* 42(11):2365–2380.

Hasle, G.; Haut, R.; Johansen, B.; and Ølberg, T. 1996. Well activity scheduling - an application of constraint reasoning. In *Artificial Intelligence in the Petroleum Industry: Symbolic and Computational Applications II*. Technip. 209–228.

Hooker, J. N. 2004. A hybrid method for planning and scheduling. *Constraints* 10:385–401.

IBM. 2010. *ILOG CPLEX Optimization Studio 12.2 documentation for ODM Enterprise*.

Jain, V., and Grossmann, I. E. 2001. Algorithms for hybrid MILP/CP models for a class of optimization problems. *INFORMS Journal on Computing* 13(4):258–276.

Méndez, C. A.; Henning, G. P.; and Cerdá, J. 2000. Optimal scheduling of batch plants satisfying multiple product orders with different due-dates. *Computers and Chemical Engineering* 24:2223–2245.

Moura, A. V.; Pereira, R. A.; and de Souza, C. C. 2008. Scheduling activities at oil wells with resource displacement. *International Transactions in Operational Research* 15(25):659–683.

Pereira, R. A.; Moura, A. V.; and de Souza, C. C. 2005. Comparative experiments with GRASP and constraint programming for the oil well drilling problem. In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms*, 328–340. Springer.

Serra, T.; Nishioka, G.; and Marcellino, F. J. M. 2011. A constraint-based scheduling of offshore well development activities with inventory management. In *Proceedings of the 43rd Brazilian Symposium on Operations Research*.

Stefansson, H.; Sigmarsdottir, S.; Jensson, P.; and Shah, N. 2011. Discrete and continuous time representations and mathematical models for large production scheduling problems: A case study from the pharmaceutical industry. *European Journal of Operational Research* 215:383–392.

PELEA: a Domain-Independent Architecture for Planning, Execution and Learning

[1] César Guzmán, [2] Vidal Alcázar, [3] David Prior
[1] Eva Onaindia, [2] Daniel Borrajo, [3] Juan Fdez-Olivares, [2] Ezequiel Quintero
[1] Universitat Politècnica de València, Camino de Vera, s/n, 46022 Valencia, Spain
[2] Universidad Carlos III de Madrid, Av. de la Universidad, 30, 28911 Leganés, Spain
[3] Universidad de Granada, Av. Hospicio, s/n, 18010 Granada, Spain
cguzman@dsic.upv.es, valcazar@inf.uc3m.es, dprior@decsai.ugr.es
onaindia@dsic.upv.es, dborrajo@ia.uc3m.es, faro@decsai.ugr.es and equinter@inf.uc3m.es

Abstract

One of the current limitations for large-scale use of planning technology in real world applications is the lack of software platforms to integrate the full spectrum of planning-related technologies: sensing, planning, executing, monitoring, re-planning and learning from past experiences. In this paper, we present PELEA, a domain-independent online planning architecture that includes state-of-the-art components for performing a wide range of (meta-)planning tasks, such as learning of control knowledge or low-level planning among many others. PELEA is conceived as a general-purpose architecture suitable for problems ranging from robotics to emergency management. PELEA is also intended to provide a rapid prototyping life-cycle for building planning applications and support planning practitioners with a highly-configurable tool.

Introduction

Automated Planning (AP) has been successfully applied to different real-world problems, such as space (Ai-Chang et al. 2004), robot control (McGann et al. 2009), or fire extinction (Fdez-Olivares et al. 2006) among many others. The process of developing a final application is an "ad-hoc" manual process that requires expertise and techniques from several fields as well as a careful definition of the underlying architecture. Most applications rely on architectures that include the functionalities required for a continuous planning, namely sensing the state, generating the problem at hand, planning , executing the plan, monitoring the execution for failures, etc. These applications are also based on replanning when needed, and, possibly, learning from the interaction to generate better models or control knowledge to improve search. However, in most applications, specifically tailored software had to be developed for the domain at hand, which usually lacks generality and reuse possibilities.

There have been some attempts, though, to design generic architectures used for different purposes. Examples can be found in space and robotics applications of platforms as Mapgen (Ai-Chang et al. 2004), APSI (Cesta et al. 2009), PRS (Georgeff and Lansky 1987), or IxTeT (Ghallab and Laruelle 1994). However, these platforms have been designed for specific problems and techniques, as timeline-based planning (Ai-Chang et al. 2004; Cesta et al. 2009; Ghallab and Laruelle 1994), hierarchical planning (Fdez-Olivares et al. 2006), or reactive controllers (Georgeff and Lansky 1987).

In this paper, we present PELEA, a domain-independent, component-based architecture able to perform planning, execution, monitoring. repairing and learning in an integrated way, in the context of PDDL-based and HTN-based planning (Alcázar et al. 2010). PELEA follows a continuous planning approach, i.e. an ongoing and dynamic process in which planning and execution are interleaved but, unlike other approaches (Myers 1999; Chien et al. 2000), it allows planning engineers to easily generate new applications by reusing and modifying the components as well as a high flexibility to compare different techniques for each module or even incorporate one's own techniques.

One particular application domain suitable for testing a continuous planning approach is *robotics* as it provides the kind of plan generation and replanning capabilities required for situated agents in highly dynamic environments. We use the general-purpose, highly-configurable architecture PELEA as an autonomous mobile robot control system. PELEA allows controlling the execution of a given task independently of the robot control platform and devices, monitoring the correct plan execution, resolving uncertainty by replanning when needed, and learning additional knowledge. To test PELEA as a robot control system we worked with the *Rovers* domain from the International Planning Competition (IPC[1]). This domain is a simplified version of the planning tasks performed by the Mars Rovers.

This paper is organized as follows. First, we present an overview of PELEA architecture. The next two sections relate the high-level and low-level planning in PELEA, respectively. The following sections describe the learning module and the goals and metrics module. Afterwards, a case study showing the features of PELEA is shown. Finally, the last section presents our conclusions.

[1] IPC: http://ipc.icaps-conference.org/

Overview of PELEA Architecture

A full description of PELEA architecture can be found in (Alcázar et al. 2010). Here, we sketch the components and main functionalities of PELEA.

PELEA architecture includes components that allow the applications to dynamically integrate planning, execution, monitoring, replanning and learning techniques. PELEA provides two main types of reasoning: high-level (mostly deliberative) and low-level (mostly reactive). This is common to most robotics applications and reflects the separation between a reactive component and a deliberative component. However, in our architecture, these are simply two planning levels. This offers two main advantages: both levels can be easily adapted to the requirements of the agent; and the differentiation allows the agent replanning at either level, which grants a greater degree of flexibility when recovering from failed executions.

Figure 1 shows a screenshot of PELEA's web interface and the current version of the architecture along with the integration of the modules. As we can see, PELEA is composed of eight modules that exchange a set of Knowledge Items (KI) during the reasoning and execution steps. We have chosen to use XML within the architecture to represent those KIs, which are: (1) *stateL*, low-level state composed of the sensory information; (2) *stateH*, abstracted high-level state translated from stateL as an aggregation or a generalization of low level information; (3) *goals*: the set of high-level goals to be achieved by the architecture; (4) *metrics*, metrics that will be used in the high-level planning process; (5) *planH*, high-level plan generated with any state-of-the-art high-level planner (this is a configurable parameter in PELEA); actions in planH can also be the goals for the low-level planner (in case we want the low-level planner to act as a dynamic translation mechanism for high-level actions); (6) *planL*, low-level plan as a set of operational actions resulting from the low-level planning that are directly executable in the environment; (7) *domainH*, definition of actions for the high-level planning; (8) *domainL*, definition of behaviors (skills) for the low-level planning learning examples; (9) *heuristics*, in different forms (control rules, policies, cases, macro-actions, etc.) allow the planner to improve the efficiency in solving future planning episodes; and (10) *info monitor*, meta knowledge on the plan that helps perform the plan monitoring (for instance, the generation time of a literal).

The high-level knowledge describes general information, actions in terms of its preconditions and effects, and typically represents an abstraction of the real problem. High-level knowledge is concerned with the description of the high-level domain, problems, goals and metrics, and they are required for the purpose of planning sequences of actions, and for the modifications of these sequences (repair or replanning). We use PDDL to represent this information. However, high-level knowledge descriptions are rarely directly executable, if ever, and they must be complemented by the low-level knowledge, which specifies how the operations are actually performed in terms of continuous change, sensors and actuators. Low-level knowledge describes the more basic actions in the simulated world, and it is typically concerned with specific rather than general functions, and how they operate. Now, the main components of PELEA are described.

Execution Module. The starting point of PELEA is the initialization of the Execution Module to capture the current problem state (*stateL*). This module also receives a high/low-level domain, and a problem. The Execution Module keeps only the static part of the initial state, given that the dynamic part, *stateL*, is read from the environment through sensors. The environment is either a hardware device, a software application, a software simulator, or a user. The Execution Module is in charge of receiving the new *stateL* and sending out the low-level actions (*planL*) that have to be executed at each step to the actuators.

Monitoring Module. Both the problem and the domain definitions and, optionally, a metric, are sent to the Monitoring Module to obtain a high-level plan (*planH*). Then, *planH* is translated into a low-level plan (*planL*) whose actions are finally sent to the Execution Module. In PELEA, it is not necessary to work at the two knowledge levels. One can just work at the high-level, so that converting knowledge from high-level into low-level with the LowToHigh module or using the Low-level planner module are not required. The Monitoring Module calls the Decision Support, which in turn calls the High-level replanner, to obtain *planH*. If the low level is being used, *planH* is converted into *planL*; otherwise, (some) actions in *planH* are directly sent to the Execution Module. Once the actions are executed, the Monitoring Module receives the necessary knowledge (current state, problem and domain) from the Execution Module and it starts the plan monitoring process. The first step is to check whether the problem goals are already achieved in the received state (*goalsL* and *goalsH* in case we are dealing with the two processes). If so, the plan execution finishes; otherwise, the Monitoring Module checks whether the received state matches the expected state or not and determines the existence or lack of a plan failure.

Low-level planner. The Monitoring Module, with the help of the Low-level planner module, generates a set of executable low-level actions (*planL*). An example of low-level knowledge would be "the coordinates of a robot" or "degrees of motion of a robot arm". If the Low-level planner module is not used, the Monitoring Module assumes that actions in *planH* are executable, and they are directly sent to the Execution Module.

Decision Support Module. It selects the variables to be observed by the Monitoring Module during the plan monitoring, and takes the decision of repairing or re-planning through an Anytime Plan-Adaptation approach (Garrido, Guzman, and Onaindia 2010) when the Monitoring detects a failure in the plan monitoring. It also communicates the Monitoring Module with the High-level replanner Module and retrieves training instances from the execution and the plans to be sent to the Learning module.

High-level replanner. It receives a problem and a high-level domain (*domainH*) and generates a high-level plan (*planH*). This module is also invoked when the Decision Support has to fix (repair/replan) a plan. In this latter case, the initial state of the problem will be the current observed

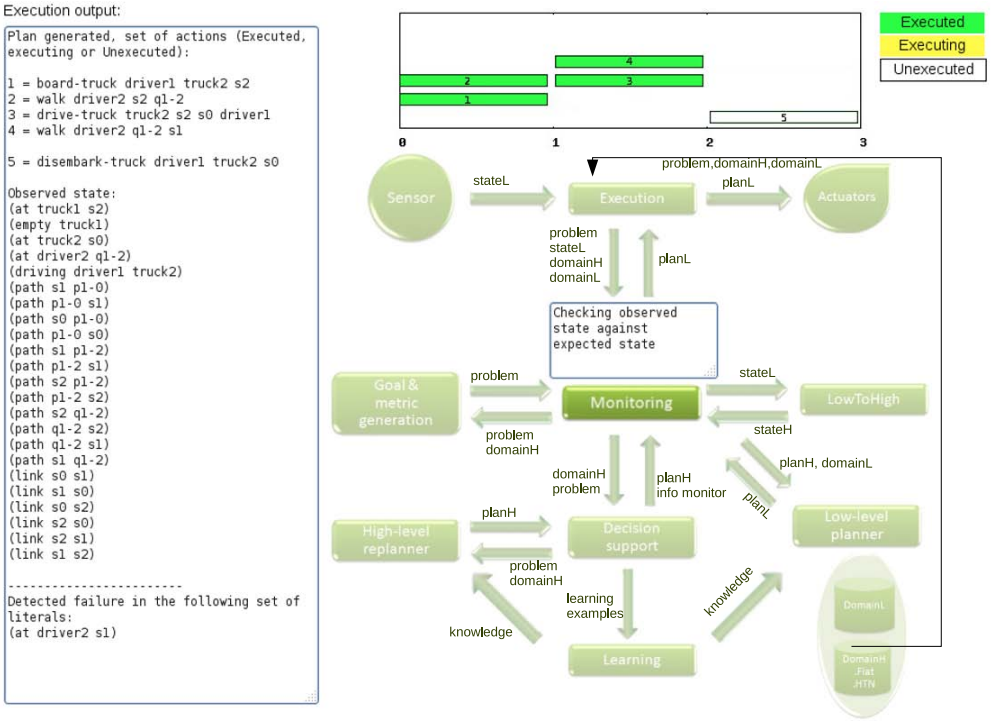


Figure 1: Screenshot of PELEA's web interface showing the architecture of the system. It shows the execution of a simple problem in the Driverlog domain.

state. Several planners have been successfully used for this module: LPG-TD (Gerevini, Saetti, and Serina 2003), SGPLAN (Hsu et al. 2007), CRIKEY (Coles et al. 2009) and TFD (Eyerich, Mattmller, and Rger 2009).

Learning Module. It infers knowledge from a training set sent by the Decision support module. The knowledge can be used either to modify the domain planning model or to improve the planning process (heuristics). Apart from the different levels of reasoning, PELEA can also learn from past executions and reason about the current problem to improve its efficiency.

The components run as separate processes and communicate through sockets. The inputs are defined by either the PDDL or HTN domain/problem specification at the high level. The knowledge exchanged among components follows the domain-independence principle with domain-independent APIs (through XML). Here lies the generality of PELEA; one can exchange a component and PELEA will continue working as it is, maintaining the XML APIs and their semantics, which are the standard ones in planning: actions, goals, states and plans.

Plan Monitoring and Decision Support

The *info monitor* parameter, provided by the Decision Support Module, comprises the information that needs to be monitored to guarantee a successful plan execution. Specifically, it includes: i) the variables to be monitored, i.e. those that are directly related to the plan, ii) the time at which the variable is generated, and the earliest and latest time at which the variable will be used, respectively; and iii) the value range for each variable, denoting the set of correct values that the variables can take on. The Decision Support Module computes the variables to be monitored through an extension of the goal regression method proposed in (Fritz and McIlraith 2007), which is inspired by the mechanism used in triangle table defined in (Fikes, Hart, and Nilsson 1972). This mechanism is only used so far to monitor the high-level information.

The Monitoring Module receives *planH* and the *info monitor* parameter and sends a set of executable actions from *planH* at a time instant t to the Execution. For example, in figure 2, at $t = 0.0003$ actions a_1 and a_2 can be executed in parallel. The Monitoring sends these two actions to the Execution and requests the execution state at the time in which variables are generated and used, most typically at the end of the execution of the actions. In this example, the Monitoring, with the help of the Execution, senses the dynamic state variables from the environment at $t = 2.0003$.

Once the information of the observed state is received by the Monitoring at time t, it checks the values of the variables are within the value range specified in *info monitor* parameter. If so, the Monitoring continues with the plan execution, sending the next set of actions to the Execution (action a_3 in figure 2). Otherwise, a discrepancy between the expected

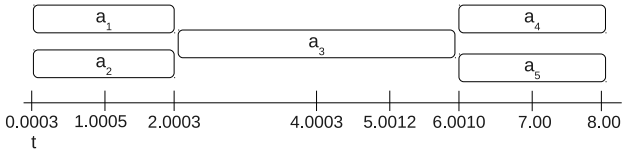


Figure 2: Example of a parallel plan.

and the observed state is found, in which case the anomaly is reported to the Decision Support, which determines whether the discrepancy is relevant to the plan execution or not. At this point, if reactivity is needed, the low-level planner is invoked to find the most immediate actions as this module typically stores predefined policies or courses of actions designed to reach particular goals. In the anomaly entails the plan is no longer executable, the Decision Support is called to take a decision between repairing the current plan or replanning from scratch.

The decision between repairing or replanning is done via the application of a regressed goal-state heuristic (Garrido, Guzman, and Onaindia 2010). A regressed goal state is a tuple of the form $GS = \langle L, t \rangle$ where L is the set of atoms, i.e. values of the state variables, and t is the time of GS, which usually coincides with the start time of one action (sequential planning) or more than one action (parallel planning). The heuristic estimates the best GS according to parameters as the cost or stability of the estimated plan. Then a new problem from S_0 to the selected regressed goal state is generated and the planner is invoked. Note that the first GS is the one from which the whole original plan can be reused; the subsequent goal states represent reachable states from which to reuse ever decreasing parts of the original plan; and the final GS entails no reuse of the plan at all.

Temporal plan monitoring

When monitoring a temporal plan, the classical definition of a regressed goal state, GS, to guarantee the executability of the plan tail from t is no longer sufficient. A *temporal regressed goal state* has now to include actions concurrent with the state variables at t, and the timing of those actions relative to the variables (Haslum 2006). That is, when computing $GS = \langle L, t \rangle$, the algorithm must take into account the actions that are being executed at time t. We will call these actions *ongoing actions*.

A temporal regressed goal state is a tuple $GS = \langle L, t, A \rangle$ where L is the values that the variables should take on at t and $A = \{(a_1, \delta_1), \ldots, (a_n, \delta_n)\}$ is the set of ongoing actions, a set of actions a_i with time intervals δ_i, meaning that each action (a_i, δ_i) in A has started δ_i time units earlier. In other words, a plan achieves the temporal regressed state GS iff the plan achieves L at t and schedules action a_i at time $t - \delta_i$ for each $(a_i, \delta_i) \in A$.

For example, consider the temporal regressed goal state $GS_1 = \langle Ls_1, 5.0012, \{(a_3, 3)\} \rangle$ in figure 3 (a sketch of a plan from a rovers problem). Since $t = 5.0012$ is the starting point of the action a_4, the preconditions of a_4 are subgoals (variables) to be achieved by t. Additionally, the action that achieves those conditions must be compatible with the on-

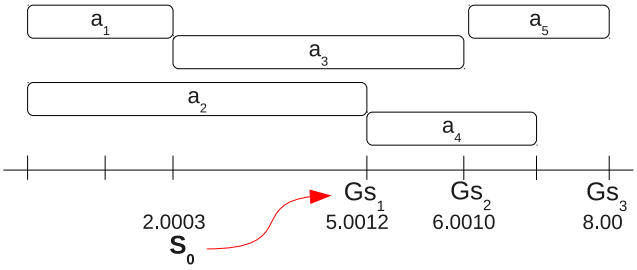


Figure 3: Temporal plan example - *planH* solution.

going action a_3, which starts 3 time units earlier. Ongoing actions are automatically computed by the Decision Support and encoded as *special PDDL actions* in the new planning domain file. Particularly, an ongoing action a_i of a temporal regressed state GS_j has the same specification as the original action except that the `at end` effects are ignored and that the duration is δ_i. The `at start` effects of the original action are included as variables in GS_j.

Analogously to the temporal regressed goal states, there may be actions that are actually being executed at the current time in the observed state (S_0). In this case, the ongoing actions have already been executed for a certain time so it is only necessary to execute them during the remaining time up to the completion of the action. See for example figure 3 where two time units of action a_2 have already been executed at S_0. Now, the current observed state is specified as follows: $S_0 = \langle L, t, A \rangle$, where L is the observed variables, t is the current time and $A = \{(a_1, \sigma_1), \ldots, (a_n, \sigma_n)\}$ is the set of ongoing actions at t, a set of actions a_i with time intervals σ_i, meaning that each action (a_i, σ_i) in A remains σ_i time units to complete. In other words, a plan from S_0 must schedule action a_i at current time t with a duration σ_i for each $(a_i, \sigma_i) \in A$. Ongoing actions of the observed state are encoded as *special PDDL actions* with: i) preconditions equal the `overall` preconditions and `at start` effects of the original action, ii) effects equal the `at end` effects of the original action and iii) a duration of σ_i time units.

Replanning *vs.* Plan Repair The Decision Support is capable of deciding between replanning or repairing in a timely fashion. It uses an algorithm with anytime capabilities whereby a first solution plan is rapidly returned, and the solution quality may improve if the algorithm is allowed to run longer (Garrido, Guzman, and Onaindia 2010). The heuristic takes a balanced response between metric (cost, makespan) and plan stability (part of the original plan that can be reused in the new solution plan) (Fox et al. 2006). Plan stability is one of the principal reasons for claiming the preference of plan repair over the alternative of replanning.

The heuristic estimates an *approximate plan* Π_{replan} (a plan from S_0 to the GS_n discarding the whole original plan), and a plan Π_{repair} (a plan from S_0 to the GS_1 keeping the whole original plan). If $cost(\Pi_{replan}) < cost(\Pi_{repair})$ or $stability(\Pi_{replan}) > stability(\Pi_{repair})$ then replanning is chosen as the preferred option. Otherwise, the algorithm analyzes the cost and stability of the subsequent regressed goal

states ($GS_2, \ldots, GS_{n-1}$) and maintains the best regressed goal state computed so far until time expires. Once a goal state GS_i is selected, the High-level replanner is invoked with the initial state S_0 and goal state GS_i. The returned plan is concatenated with the plan tail of the original plan taking into account the causal links and time constraints.

Closer to the Real World: Low Level

Actions in low-level plans (*planL*) are atomic actions that are executed directly in the environment. The low-level reasoning components are often required to come up with a solution in a short time. In this section, we present in detail the PELEA modules that implement the low level behaviour.

Execution Module

The Execution Module deals with the communication between PELEA and the environment, which can be represented by a simulator, a hardware device (robot), a third-party software, or a user. Currently, the Execution Module works as a wrapper over anything external to PELEA, tackling with low-level details that depend on the kind of environment PELEA is working with. Issues like communication protocols and data formats are responsibility of the Execution Module.

Low-level states (*stateL*) are sent to the Monitoring Module upon request. Similarly, actions in *planL* are sent to the actuators when they are received from the Monitoring. This is usually interleaved, although PELEA may ask asynchronously for *stateL* without sending any action to monitor the execution. Currently, the Execution Module provides integration with the following environments:

- MDPSim: PPDDL (Younes and Littman 2004) simulator employed in the past probabilistic tracks of the International Planning Competition (IPC)[2] which generates states stochastically as actions are received. It works with a probabilistic version of the regular PDDL domains, PPDDL, in which the effects of the actions depend on a series of probabilities.
- In-house temporal probabilistic simulator: we have built a new simulator that is able to work with temporal probabilistic domains. The domain definition is similar to the temporal version of PDDL, augmented with probabilistic effects as in PPDDL.
- Stage/Player (Gerkey, Vaughan, and Howard 2003): freeware platform for robot independent control.
- Microsoft Robotics Studio: a robot independent platform similar to Player.
- Freeware software suite for robotics applications, research and education, which also offers the possibility to simulate various kinds of robots.
- AIlife: open platform for simulating social and emotion oriented games.
- Planning framework designed to simulate and solve real-life logistic problems.

[2]Except for the last one held on 2011.

Low-Level Planner

The Low-Level Planner generates *planL* composed of atomic actions that can be directly executed in the environment. It receives *stateL* and a high-level action (the next one from the current *planH*). This high-level action conforms the low-level goal. The low-level actions of the *planL* generated by the Low-level Planner are then sent to the Execution Module. Thus, high-level plans can be decomposed into several low-level actions, keeping the reasoning at both levels distinct. The Low-level planner does not have to be a regular PDDL-based planner. We have implemented several types of low-level planners:

- HighToLow translators, that simply decompose a high-level action into low-level ones with no reasoning. They can be seen as small programs that take as input a high-level action and the low-level state and generate as output a set of low-level actions
- A policy, either learned or manually created. For instance, based on a states-actions table (as in most reinforcement learning approaches).

Suppose we have a domain in which a robot can move along a building and turn around to change its orientation. A high-level plan may contain the *(move location1 location2)* action as the current one, so it is sent to the Low-level planner. The other input would be *stateL* that includes information on the *x* and *y* coordinates of the robot position and its orientation. Then, the Low-level planner solves the path-planning problem and returns a sequence of atomic actions to be sent to the robot; for instance, the sequence *(advance 10)*, *(turn 45 right)*, *(advance 5)*.

LowToHigh Module

The task of the LowToHigh Module is to translate a *stateL* into a *stateH*. In most cases it is just a mapping function between the low and the high level, although more complex functionalities can also be implemented. The requirement of being able to generate a *stateH* from a *stateL* is justified by the necessity of monitoring at both low and high levels. Thus, replanning and repairing can be performed during the high-level execution as well.

Learning in PELEA

The Learning Module is in charge of inferring knowledge from the training data sets sent by the Decision Support module. So far, machine learning techniques have been used to improve the planning process by learning domain models, low-level policies and heuristics. We will shortly describe them here as a summary of the kinds of techniques that PELEA integrates into its core, since most of them have already been published elsewhere.

Learning domain models

The generation of accurate robot control PDDL or PPDDL models for planning is complex. To alleviate this, machine learning has been used to support model generation. As an example, a relational learning approach was developed as

part of the Learning Module of PELEA to learn more accurate robot-action execution durations than the originally modeled ones (Quintero et al. 2011). PELEA starts with a deterministic version of a robotics domain and then executes actions and observes the results of those executions in terms of the function whose effects we want to learn (e.g. navigate). We used TILDE (Blockeel and Raedt 1998) to learn the duration of actions of the Rovers domain using regression trees. Finally, the duration models were later compiled into the PDDL domain specification. The experiments were performed using a Pioneer robot that traveled through different terrain types, generating the learning examples from the type of terrain and the time it took the robot to move from one waypoint to another.

Learning low-level policies

Also, as part of the Learning Module, domain-specific learning techniques can be used at the low-level. As a proof of concept, a policy made to substitute the low-level planner was inferred using reinforcement learning. This was done for the case study described later, in which training examples of the form $\langle state, action, state, reinforcement \rangle$ were employed to create a state-action table.

Learning heuristics

The Learning Module is also able to learn heuristics for improving the planner's efficiency. Relational decision trees were used again to learn how to generate look-ahead states to improve forward search algorithms (De la Rosa et al. 2011). This was done by creating training examples that take into account the helpful actions of previous heuristic evaluations of states belonging to smaller problems of the same domain. The learning system is domain-independent, but planner-dependent, so in the future we would like to include other planner-independent techniques.

Goals and Metrics Generation

The Goal&metric generation Module is designed to automatically select the new goals and metrics to be used according to the current state of the execution. A common use of this module is for oversubscription problems, where not all goals can be satisfied. This problem is generally solved by choosing some goals and discarding others either online or offline. An algorithm that computes an estimation of the cost of achieving a goal from every other goal was implemented so a set that maximizes the number of goals that will be likely achieved can be easily found (García-Olaya, de la Rosa, and Borrajo 2011).

A Case Study: Rovers domain

In this section, we show an instance of the Rovers domain which replicates the expected behavior of the autonomous explorers sent to Mars by NASA in past experiments. It was simulated both by using Player/Stage (Gerkey, Vaughan, and Howard 2003), and two real robots, two Pioneers P3-DX,

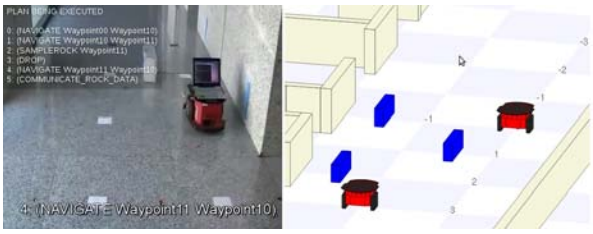


Figure 4: Execution of PELEA: real robot and simulator.

which interacted in a recreated physical space that represented the Mars environment [3].

Both the robot and the simulation were managed using Player/Stage, so there was a single implementation for both cases. The actuators/sensors management was implemented in the Execution Module as a set of basic control skills. The communication was performed using sockets, acting Player/Stage as the server and PELEA as the client. Both the Low-Level Planner and the LowToHigh Module were implemented as an ad-hoc translator adapted for the example.

Description of the case study

We have used the Rovers domain introduced in the IPC in 2002. In this domain, a collection of Rovers navigate on Mars' surface, looking for samples/images data which they should communicate back to Earth. In this case, the STRIPS version of the domain was used for the sake of simplicity. In the executed instance two different Rovers are used, and the goal is to communicate a set of samples taken from rocks in the environment. In the trace example we will refer to one of these two Rovers, named Curiosity. During the execution, both the physical Pioneer robots and the simulator provided by Player/Stage are used. Figure 4 shows the real robot during execution and a screenshot of the simulator at that instant.

Trace of execution

Now, we show the execution of the action `(navigate curiosity waypoint00 waypoint01)`. First, *stateL* is requested to the Execution Module. Curiosity will retrieve its current *stateL*, indicating that the position of Curiosity is $(x = 0.00, y = 0.00, z = -1.50)$, no bumper detected a collision, and no object is detected to be close by its sonar ring.

Once *stateL* is retrieved, the PELEA flow continues. *stateL* is translated into *stateH* and checked by the Monitoring module. After the monitoring process is done, the Low-level planner is executed in order to generate the next *planL*. The high-level action being executed (in our example : `(navigate curiosity waypoint00 waypoint01)`) is sent to the Low-Level Planner and the corresponding *planL* is generated.

For the case of our `(navigate curiosity waypoint00 waypoint01)` action, the corresponding

[3] Currently, we have already uploaded PELEA with ROS into two humanoid robots, NAOs, and we are using PELEA to control them for simple tasks

```
<plan id="navigate">
   <action-plan name="turnleft">
      <term name="robot" value="curiosity"/>
   </action-plan>
   <action-plan name="movetowardsy">
      <term name="robot" value="curiosity"/>
      <term name="y" value="0.00"/>
   </action-plan>
</plan>
```

Figure 5: *planL* returned

planL for the actuators is shown in Figure 5. Curiosity has to change its orientation turning left. And then it has to move towards the position $y = 1.0$. Now that the *planL* has been generated, the robot can execute the set of commands to achieve the complete level action. First, Curiosity turns left and then moves towards position $y = 1.00$ (PELEA sends a command `movetowardsy` as the one shown in Figure 5). Once the action is executed, the resulting *stateL* is returned by the EM. This state is translated into *stateH* that should look like the fragment on Figure 6.

```
<atom predicate="at">
  <term name="curiosity"/>
  <term name="waypoint01"/>
</atom>
<atom predicate="full">
  <term name="curiositystore"/>
</atom>
<atom predicate="calibrated">
  <term name="logitechsph"/>
  <term name="curiosity"/>
</atom>
...
```

Figure 6: *stateH*

Related Work

In the late 80's and beginning of the 90's there was a profusion of architectures for autonomous mobile robot systems which heavily drew upon the popular three-layer SPA (sensing-planning-acting) architecture. Most of these early approaches (RAP (Firby 1987), PRS (Georgeff and Lansky 1987), SHAPIRA (Saffiotti, Konolige, and Ruspini 1995)) developed reactive behaviors, fixed pre-compiled patterns of actions which are selected depending on the actual situation of the executor. Other architectures smoothly integrate planning and reacting running asynchronously a classical AI planner in conjunction with a reactive control mechanism (Gat 1992; Hayes-Roth 1995). An alternative direction is to generate plans to solve the entire planning task and monitor their execution afterwards, resorting to re-planning or plan repairing in case of execution failures (Currie and Tate 1991). The Continuous Planning and Execution Framework CPEF (Myers 1999) is an asynchronously working architecture that interleaves planning and execution. CPEF can generate plans to arbitrary levels of refinement and then be manipulated at runtime by the executor component. PELEA gathers many of the features of the aforementioned architectures, namely reactive execution, continuous planning approach, re-planning and repairing techniques but it also features a learning module and the ability to change goals as new information is acquired during plan execution (Goal&metric generation module).

PELEA also has some features in common with the Task Control Architecture TCA (Simmons 1992). TCA was developed to handle robot control problems and was specifically tested with Ambler, a robot designed for planetary explorations, which required a rather deliberative architecture. TCA provides a general framework of hierarchical task decomposition augmented with reactive behaviors. Since TCA was specifically thought of as a high-level robot operating system, it allows for the definition of task-specific modules. This contrasts with the more general PELEA behavior, which has been designed to tackle any type of planning-execution problem, from the most proactive to the most reactive domains. PELEA also uses PDDL and HTN definitions that are currently the standards in deliberative planning, so it can more easily be used by planning practitioners.

IDEA (Intelligent Distributed Execution Architecture) (Aschwanden et al. 2006) is a real-time architecture that departs from the three-layer SPA architecture and proposes instead to unify deliberation and execution under a single planning technology and model representation. Like T-Rex, IDEA is composed of self-contained planning systems, each with a deliberation latency and planning horizon. IDEA uses XIDDL, a XML encoding of IDEA domain definition language, but it does not allow for PDDL or HTN-based model representation. Additionally, this unified view that permits the planner to be embedded within the executor usually allows only for a strict and controlled interleaving of the planning and execution phases (Vidal and Nareyek 2011), making it difficult to have a general-purpose planner for different types of executor systems.

As a whole, PELEA boosts flexibility, modularity, generality and interoperability. PELEA allows practitioners to replace (and reuse) any module of the architecture as long as the new module is able to read the corresponding XML input file, thus requiring much less effort to easily generate new interleaved planning-and-execution applications.

Conclusions

In this paper, we have introduced a domain-independent architecture, PELEA, that integrates planning related processes, such as sensing, planning, execution, monitoring, replanning and learning. We have shown examples of the two levels of reasoning in a temporal and no-temporal domains. High level as in a regular automated planning task; and low level, composed by atomics actions that are executed directly in the environment. PELEA is conceived as a flexible and modular architecture that can accommodate state of the art techniques that are currently used in the overall process of planning. This kind of architectures will be a key resource to build new planning applications, where knowledge engineers will define some of the components, parameterize others, and reuse most of the available ones. This will allow engineers to easily and rapidly develop applications that incorporate planning capabilities. We believe this kind of architecture fills part of the technological gap between planning techniques and applications.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. MAPGEN: Mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Alcázar, V.; Guzmán, C.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindia, E. 2010. PELEA: Planning, learning and execution architecture. In *PlanSIG'10*, 17–24.

Aschwanden, P.; Baskaran, V.; Bernardini, S.; C. Fry, M. M.; Muscettola, N.; Plaunt, C.; Rijsman, D.; and Tompkins, P. 2006. Model-unified planning and execution for distributed autonomous system control. In *Workshop on Spacecraft Autonomy: Using AI to Expand Human Space Exploration*. AAAI Press.

Blockeel, H., and Raedt, L. D. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA*.

Chien, S. A.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *AIPS*, 300–307.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence Journal* 173(1):1–44.

Currie, K., and Tate, A. 1991. O-Plan: the open planning architecture. *Artificial Intelligence* 52(1):49–86.

De la Rosa, T.; Jiménez, S.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research* 40:767–813.

Eyerich, P.; Mattmller, R.; and Rger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proc. ICAPS 2009*.

Fdez-Olivares, J.; Castillo, L.; García-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together. experiences in SIADEX. In *Proc. ICAPS 2006*. Awarded as the Best Application Paper of this edition.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

Firby, R. J. 1987. An investigation into reactive planning in complex domains. In *AAAI*, 202–206.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proc. ICAPS 2006*, 212–221.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *ICAPS*, 144–151.

García-Olaya, A.; de la Rosa, T.; and Borrajo, D. 2011. Using relaxed plan heuristic to select goals in oversubscription planning problems. In *Advances in Artificial Intelligence*.

Garrido, A.; Guzman, C.; and Onaindia, E. 2010. Anytime plan-adaptation for continuous planning. In *PlanSIG'10*, 62–69.

Gat, E. 1992. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAI*, 809–815.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of AAAI-87 Sixth National Conference on Artificial Intelligence*, 677–682.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research* 20:239–290.

Gerkey, B.; Vaughan, R.; and Howard, A. 2003. The player/stage project: Tools for multi-robot and distributed sensor systems. In *ICAR 2003*.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *2nd International Conference on AI Planning Systems*.

Haslum, P. 2006. *Admissible Heuristics for Automated Planning*. Ph.D. Dissertation, Linkopings Universitet.

Hayes-Roth, B. 1995. An architecture for adaptive intelligent systems. *Artif. Intell.* 72(1-2):329–365.

Hsu, C.-W.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. Constraint partitioning for solving planning problems with trajectory constraints and goal preferences. In *IJCAI'07*.

McGann, C.; Py, F.; Rajan, K.; and Olaya, A. G. 2009. Integrated planning and execution for robotic exploration. In *Procs. of International Workshop on Hybrid Control of Autonomous Systems*.

Myers, K. L. 1999. Cpef: A continuous planning and execution framework. *AI Magazine* 20(4):63–69.

Quintero, E.; Alcázar, V.; Borrajo, D.; Fdez-Olivares, J.; Fernndez, F.; Ángel García-Olaya; Guzmán, C.; Onaindia, E.; and Prior, D. 2011. Autonomous mobile robot control and learning with pelea architecture. In *PAMR'11*, 51–56. AAAI Press.

Saffiotti, A.; Konolige, K.; and Ruspini, E. H. 1995. A multivalued logic approach to integrating planning and control. *Artif. Intell.* 76(1-2):481–526.

Simmons, R. 1992. Concurrent planning and execution for autonomous robots. In *IEEE International Conference on Robotics and Automation*, 46–50.

Vidal, E. C. J. E., and Nareyek, A. 2011. A real-time concurrent planning and execution framework for automated story planning for games. In *Intelligent Narrative Technologies*.

Younes, H. L. S., and Littman, M. L. 2004. PPDDL1.0: An extension to pddl for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-167.

Digital Cityscapes: Challenges and Opportunities for Planning & Scheduling

Ming C. Lin and Dinesh Manocha
University of North Carolina at Chapel Hill

Abstract

In this position paper, we examine the algorithmic and computational challenges in the applications of planning and scheduling for real-time modeling and simulation of *digital cityscapes*. We outline the areas of research challenges in *digital cityscapes* that can benefit from more advanced planning and scheduling algorithms and techniques. We briefly survey some of our recent progress as part of our early attempt in adopting multi-agent planning and scheduling for digital cityscapes and highlight some of the remaining challenges.

1 Introduction

With industrial revolution, recent economic and social development, increasingly more people are leaving rural areas and migrating to cities, thereby leading to rapid urbanization of the world population in the last century. Today, more than 50% of the global population live in urban areas, with the figure projected to rise to 60% by 2030[1]. Given the ubiquitous urban development across all advanced and developing countries, modeling and simulation of cityscapes is clearly emerging as an important topic for city planning and urban development that require interactive visualization to evaluate various alternatives and options for design and planning. The scale and complexity of the problem demand a new set of algorithms and methodologies for visualizing rich, intricate, and dynamic urban landscapes with constant flows of crowds and traffic.

Numerous efforts have been devoted in acquiring and visualizing "urbanscape". Over the last decade, there has been considerable progress on multiple fronts: acquisition of imagery and 3D models using improved sensing technologies, real-time rendering, and procedural modeling. For example, aerial imagery of most cities is used in Google Earth and Microsoft Virtual Earth. The problem of reconstructing 3D geometric models from videos and scanners has been an active area of research in computer vision and related areas. Similarly, many efficient techniques have been proposed to stream the imagery and geometric data over internet and display them at real-time rates on high end workstations or handheld devices. However, all these efforts are limited to capturing, displaying, or modeling predominantly static models of urbanscapes and do not include dynamic elements, such as crowds or traffic. In many aspects, the realism of models shown in Google Earth or Microsoft Virtual Earth is lacking due to the absence of dynamic behaviors.

Figure 1: *An example of simulated crowds at Shibuya crossing in Japan*

In addition to high-rise buildings and architectural scenes on city landscapes, moving pedestrians and vehicle traffic are an integral part of any metropolitan region, yet they have not received sufficient attention. Aggregates of numerous entities, such as a group of people and fleet of vehicles, form complex systems that exhibit interesting biological, social, cultural, and spatial patterns observed in nature and in society. Modeling of the collective behaviors remains an open research challenge in artificial intelligence, computer vision, architecture, physics, psychology, social sciences, and civil and traffic engineering, as complex systems often exhibit distinct characteristics, such as emergent behaviors, self-organization, and pattern formation, due to multi-scale interactions among individuals and groups of individuals, despite of decades of observation and studies.

2 Research Challenges

The challenges in real-time modeling and simulation of digital cityscape stem from its extremely large scale, i.e. in the range of hundreds of thousands or even millions, crowds and vehicle traffic commonly encountered in metropolitan areas across the globe. We refer to such a physically vast scale of computational challenges as "metropolitan scale." Below we briefly list a few problems in realizing this vision and provide pointers to some recent progress toward this goal:

[1] "World Urbanization Prospects" by *United Nations* Population Division, Department of Economic and Social Affairs, 2005.

Figure 2: *An example of reconstructed traffic in an European cityscape*

- **Modeling of intricate pedestrian dynamics that leads to better understanding of complex crowd pheonomena**: Recently we have developed a new trajectory planning algorithm for virtual humans. Our approach focuses on implicit cooperation between multiple virtual agents in order to share the work of avoiding collisions with each other. Specifically, we extend recent work on multi-robot planning to better model how humans avoid collisions by introducing new parameters that model human traits, such as reaction time and biomechanical limitations. We validate this new model based on data of real humans walking captured by the Locanthrope project. Extending such approach to many thousands or millions of people in a large crowd remains a significant challenge. See:

  `http://gamma.cs.unc.edu/RCAP`

  `http://gamma.cs.unc.edu/PLE`

- **Real-time reconstruction metropolitan-scale traffic flows given discrete temporal-spatial sensor data**: We introduce a novel concept, Virtualized Traffic, to visualize reconstructed continuous traffic flows from traffic sensor datas. Given the positions of each car at two recorded locations on a highway and the corresponding time instances, our approach can recreate the traffic flows (i.e. the dynamic motions of multiple cars over time) in between the two locations using a priority-based scheduling scheme for multiple agents. Our algorithm is applicable to high-density traffic on highways with a small number of lanes and takes into account the geometric, kinematic, and dynamic constraints on the cars. Although our framework can process a continuous stream of input data in real time by reducing the search space for planning, extending such approaches to a large number of lanes with finer discretization to better approximate contiuous motion makes this approach quickly intractable. More efficient techniques would be needed. See:

  `http://gamma.cs.unc.edu/TRAFFIC-RECON`

- **Data-driven personality models based on perceptual studies for simulating crowd and driver behaviors**: To generate heterogeneous crowd behaviors using personality trait theory, we adopt results of a user study to derive a mapping from crowd simulation parameters to the perceived behaviors of agents in computer-generated crowd simulations. We establish a linear mapping between simulation parameters and personality descriptors corresponding to the well-established Eysenck Three-factor personality model. Furthermore, we propose a novel two-dimensional factorization of perceived personality in crowds based on a statistical analysis of the user study results. Extension to this approach to establish dynamic mappings and factorizations for generating heterogeneous crowd behaviors in settings with external factors (such as interaction with other agents, environments, and other stress factors) would need to be considered as well. See:

  `http://gamma.cs.unc.edu/personality`

- **Applications to traffic rerouting and congestion management**: While state-of-the-art systems take into account current trafc conditions or historic trafc data, current planning approaches ignore the impact of their own plans on the future trafc conditions. We introduce a novel algorithm for self-aware route planning that uses the routes it plans for current vehicle traffic to more accurately predict future traffic conditions for subsequent cars. Our planner uses a roadmap with stochastic, time-varying traffic densities that are defined by a combination of historical data and the densities predicted by the planned routes for the cars ahead of the current trafc. We have applied our algorithm to moderate-scale traffic route planning, and demonstrated that our self-aware route planner can more accurately predict future traffic conditions, which results in a reduction of the travel time for those vehicles that use our algorithm. Extension of such planning and scheduling framework to metropolitan-scale traffic that incorporates dynamic sensing and real-time traffic prediction would introduce new challenges to planning and scheduling. See:

  `http://gamma.cs.unc.edu/TROUTE`

  Other applications including emergency response and planning, architecture and engineering design evaluation, etc. should also be investigated. In addition, validation of such techniques should be also addressed in the context of applications.

3 Conclusion

We have suggested a list of problems in developing digital cityscapes that can benefit from applications of more advanced planning and scheduling algorithms and techniques. Addressing these problems can lead to attaining plausible explanations of the behavior and motivation of individual agents (e.g. pedestrians or vehicles) and how they interact with each other under different settings, across varying scales and levels of social organizations, from individuals to groups, with applications ranging from urban planning, civil and traffic engineering, transportation system design, architectural layout, training of first-responders electronic commerce, to education and entertainment.

Planning Task Validation

M. Viviane Menezes and **Leliane N. de Barros**
Department of Computer Science
Institute of Mathematics and Statistics
University of São Paulo, Brazil
{mariavm,leliane}@ime.usp.br

Silvio do Lago Pereira
Department of Information Technology
FATEC-SP\CEETEPS
São Paulo, Brazil
slago@pesquisador.cnpq.br

Abstract

A planning agent should be able to deal with different situations and goals, while performing a task. However, in a real-world planning application, the agent can fail in many different ways. In special, failure to find a plan is an important type of situation to reason about both, for debugging planning systems and for planning task validation (including the validation of the planning domain model and the task specification). Unfortunately, the validation of planning tasks is still a poorly developed area and has few tools providing this kind of functionality. In this paper we are interested in discussing existing solutions for the planning task validation problem and present promising ideas for the development of new validation tools.

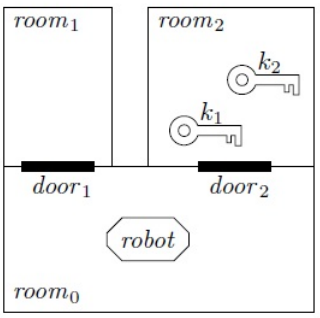
Introduction

A planning agent should be able to deal with different situations while performing a task. However, in a real-world planning application, the agent can fail in many different ways, e.g.:

- **Fail to execute a plan.** This can happen since the agent is not perfect or because the environment has changed, e.g., the agent can fail to take an object or drop a holding object (which can be repaired by executing the same action or a new one that takes the object from the floor). In the case the environment has changed in such way that no plan repair can be performed, replanning is usually possible, otherwise, the planner fails to find a new plan.
- **Fail to find a plan.** In this case, a small change in the planning task would allow the agent to find a plan.

The first type of failure is related to plan repair and replanning research area and the second one to planning system debugging and planning task validation, which includes validating the planning goal, initial state (also including the domain objects) and domain action descriptions. While in the plan repair and replanning research areas there are many works (Fox et al. 2006; Yoon, Fern, and Givan 2007; Brenner and Nebel 2009), unfortunately, the planning task validation area is still poorly developed and has few tools providing this kind of functionality (Penix, Pecheur, and Havelund 1998; Howey, Long, and Fox 2004; Göbelbecker et al. 2010). That is because it is a complex problem: domain models and planning tasks are often composed of a large number of conditional constraints and dependencies. In fact, the modeling and the validation of a new critical task (e.g., a mission planning application) require an intensive interaction with human expert. A good supporting system for modeling and validating a new planning task should be a mixed-initiative system composed of a number of supporting tools. In this paper, we are interested in discussing solutions for the *planning task validation* problem.

Consider the planning domain named *Keys* where a robot can navigate among rooms opening doors using specific keys. Figure 1 shows a problem in this domain containing: a robot, three rooms, two doors and two keys. In this example, key_1 opens $door_1$ and key_2 opens $door_2$. The robot goal is to reach $room_1$. Considering the initial state depicted in Figure 1, it is impossible for the robot to reach $room_1$, since the $door_1$ is locked and both keys are locked inside the $room_2$. To fix that, we can change the initial state of the planning task by: (i) locating the robot at $room_2$ or; (ii) putting a new key to open $room_2$ in $room_0$ or; (iii) creating a new action to call a floor guard to open the door.




```
(:objects
  room0 room1 room2 - room
  door1 door2 - door
  key1 key2 - key)

(:init
  (= (robot_pos) room0)
  (= (key_pos key1) room2)
  (= (key_pos key2) room1)
  (connects room0 room1 door1)
  (connects room1 room0 door1)
  (connects room0 room2 door2)
  (connects room2 room0 door2)
  (= (key_opens key1) door1)
  (= (key_opens key2) door2)
  (locked door1)
  (locked door2))

(:goal (= (robot_pos) room1))
```

Figure 1: Unsolvable planning task in the *Keys* domain. The agent goal is to reach $room_1$ that is locked and whose key (key_1) is in the locked $room_2$ (Göbelbecker et al. 2010).

Göbelbecker et al. (2010) proposed a method, called in this paper as *Giving-Excuses* method, that automatically finds excuses for an unsolvable classical planning task. In

order to find excuses, the method uses two structures: the *causal graph* of the planning domain and the *domain transition graph* of each variable [1]. Based on them, the proposed method creates new actions (called *change-value* actions), that can actually change the value of a single variable. The method also creates the *add-object* actions, to add new objects in the planning task (e.g., a new key in room_0). By using an out-of-shelf planner, and considering the set of *change-value* and *add-object* actions to be applied in the initial state s_0, the method returns a new initial state (from which is possible to find a solution), i.e., an *excuse* for the planning failure.

Note that the *change-value* and *add-object* actions are only used during the process of defining a new initial state and are not considered as part of the planning domain. Thus, the *Giving-Excuses* method cannot be used to, automatically, propose changes in the set of planning domain actions.

In this paper we briefly describe the *Giving-Excuses* method, and show how it can be formalized in propositional logic (to be implemented using *Binary Decision Diagrams* (BDDs) (Bryant 1992)). One of the advantages of this formalization is that we do not need to construct and use the *causal* and *domain transition* graphs. We also discuss possible extensions of this method to allow changes in the domain action specification for planning task validation. We also resume the *Planning as Model Checking* work and discuss how to propose changes in the initial state based on these techniques, and how to modify the domain actions.

Planning Task Validation Problem

We define a *planning task validation problem* $\mathcal{V}$ as a tuple $\langle \Delta, \Gamma \rangle$ where Δ is the planning domain specification and Γ is a set of planning tasks. The validation of $\mathcal{V}$ is successful if all $\mu \in \Gamma$ can be solved, otherwise, the validating system returns a set of minimal possible changes for each planning task $\mu \in \Gamma$ that cannot be solvable and\or the set of domain actions that forces μ to be solvable. The minimal changes for a planning task can be either a change in the initial state and\or in the set of domain objects.

Changes in domain actions can be: relax (or constrain) an action precondition and relax (or constrain) an action effect.

Giving these possible changes, an user of a planning task validation system should be able to choose among them. I.e., having defined a metric to find minimal changes (possibly more than one), the system could return all the minimal changes and it is up to the user to decide for one of them, according to his preferences or beliefs.

In the next section, we describe an approach, the *Giving-Excuses* method, that is able to return minimal possible changes for a given planning task. We also discuss how we can formalize this method to implement its symbolic version using BDDs and briefly discuss how it can be modified to also return minimal possible changes for domain actions to force a planning task to have a solution.

[1] The causal graph and the domain transition graph are used by the Fast Downward planning algorithm (Helmert 2006)

move(room ?from, room ?to, door ?d)
pre: robot-pos = ?from $\wedge$ locked(?d) = $\bot$ $\wedge$
connects(?from ?to ?d) = $\top$
eff: robot-pos = ?to

Figure 2: An example of action SAS$^+$ for the Keys domain. Move the robot from a room *?from* to another room *?to* through a door *?d*.

The Giving-Excuses Method with SAS$^+$ Language

The *Giving-Excuses* method assumes that the domain is described in SAS$^+$ (Bäckström and Nebel 1995), a language to describe planning domains using multi-valued variables with finite domain. This language is used by the Fast Downward planning system (Helmert 2006), winner of the classical track of the 4th IPC, which uses hierarchical decompositions of planning tasks for computing a heuristic function, based on the causal graph of the domain.

Definition 1. *(Planning Domain SAS$^+$) The planning domain SAS$^+$ is a tuple $\Delta = \langle C_\Delta, \mathcal{V}, \mathbb{A}, \rangle$ where:*

- *C_Δ is the set of domain constant symbols;*
- *$\mathcal{V}$ is the set of fluent and predicate symbols, with associated arities and typing schemata and;*
- *$\mathbb{A}$ is the set of actions.*

Definition 2. *(Action SAS$^+$) Each action SAS$^+$ $a \in \mathbb{A}$ is composed by preconditions and effects with multivalued variables.*

Figure 2 shows the description of the action SAS$^+$ `move` from the Keys domain, wich is responsible to conduct the robot from a room to another. The *move* action has three parameters: the room where the robot is (source room), the room where the robot intends to go (destination room) and the door that connects both rooms. The action will be executed if the robot is in a source room, if the door connecting the source room with destination room is unlocked and if the rooms are connected. The effect of the action is to change the localization of the robot to the destination room.

Definition 3. *(Planning Task SAS$^+$) The planning task SAS$^+$ is a tuple $\Pi = \langle \Delta, s_0, s^* \rangle$ where:*

- *Δ is the planning domain;*
- *s_0 is the description of the initial state and;*
- *s^* is the goal specification.*

Definition 4. *(Domain of a SAS$^+$ Variable) A SAS$^+$ variable is a ground fluent $f \in \mathcal{V}$ or a ground predicate $p \in \mathcal{V}$. The domain of a ground fluent f is the set $dom(f) \subseteq \mathcal{C}$. The domain of a ground predicate p is the set $dom(p) \subseteq \{\bot, \top\}$.*

Given an unsolvable SAS$^+$ planning task, an **excuse** is a pair $\langle \mathcal{C}_\mathcal{X}, s_\mathcal{X} \rangle$ that implies a solvable task, where $\mathcal{C}_\mathcal{X}$ is a new set of objects and $s_\mathcal{X}$ a new initial state (Göbelbecker et al. 2010). The pair $\langle \mathcal{C}_\mathcal{X}, s_\mathcal{X} \rangle$ takes a view that an excuse is a counter-factual statement: the task has a solution if we add objects $\mathcal{C}_\mathcal{X}$ and have $s_\mathcal{X}$ as initial state. An excuse can

be classified as **acceptable**, **good** or **perfect**. Given two excuses $\mathcal{X} = \langle s_{\mathcal{X}}, C_{\mathcal{X}} \rangle$ and $\mathcal{X}' = \langle s_{\mathcal{X}'}, C_{\mathcal{X}'} \rangle$ we say that $\mathcal{X}$ is *at least as acceptable as* $\mathcal{X}'$ ($\mathcal{X} \preccurlyeq \mathcal{X}'$) iff $C_{\mathcal{X}} \subseteq C_{\mathcal{X}'}$ and $s_0 \setminus s_{\mathcal{X}} \subseteq s_0 \setminus s_{\mathcal{X}'}$[2]. A minimal element under the ordering $\preccurlyeq$ is called an *acceptable excuse*. Given two acceptable excuses $\mathcal{X} = \langle s_{\mathcal{X}}, C_{\mathcal{X}} \rangle$ and $\mathcal{X}' = \langle s_{\mathcal{X}'}, C_{\mathcal{X}'} \rangle$, we say that $\mathcal{X}$ is *at least as good as* $\mathcal{X}'$ ($\mathcal{X} \sqsubseteq \mathcal{X}'$) if $\mathcal{X}$ subsumes $\mathcal{X}'$, i.e., changes caused by $\mathcal{X}'$ can be explained by $\mathcal{X}$ (e.g., in the *Keys* domain $\mathcal{X}$ can lead to a state where key_2 is with the robot and $\mathcal{X}'$ can lead to a state in which $door_2$ is unlocked). Good excuses with minimal cost are called *perfect excuses*.

The *Giving-Excuses* method is able to find good excuses for an unsolvable planning task by analyzing two structures: two structures: the *causal graph* and *domain transition graphs* to the domain description.

Definition 5. *(Causal graph) The causal graph (S, E) (Helmert 2006) is a structure that captures the causal dependencies between variables of the planning domain: (i) each vertex represents a variable and (ii) an edge between two vertex represents a causal dependence between the value of these variables. An edge $(u, v) \in E$ iff $u \neq v$ and there is $a \in \mathbb{A}$ such that $u \in pre(a)$ and $v \in eff(a)$ or both $u, v \in eff(a)$. We say either u causes v or v depends on u.*

Definition 6. *(Variable Dependents) Given the causal graph (S, E), the set of variables u on which v depends is given by:*

$$dependentsOf(v) = \{u \mid \exists (u, v) \in E\}.$$

Figure 3 shows the causal graph for the *Keys* domain with multi-valued variables. Each state variable is represented by a vertex and the dependencies between the variables is represented by edges.

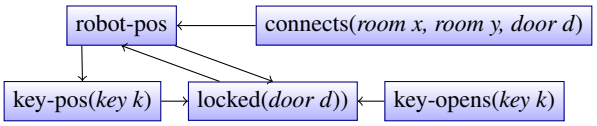


Figure 3: Causal graph for the *Keys* domain with multi-valued variables.

Definition 7. *(Domain transition graph) The domain transition graph (Helmert 2006) of a variable represents the possible ways that this variable can change its values and the necessary conditions for that change to occur.*

Figure 4 shows the domain transition graph for the multi-valued variable *robot-pos* of the *Keys* domain. In this graph, each vertex is a possible value that the variable can assume and the edges are labeled by the necessary conditions for that variable to change its value. Variable *robot-pos* can assume three possible values ($room_0$, $room_1$ and $room_2$) and its value can be changed if the specific door is unlocked and the related rooms are connected.

Among all the possible values that a SAS$^+$ variable v can assume ($dom(v)$), there are those *relevant* to the agent to reach its goal. The set of these values is called *relevant domain* of a variable.

[2]The symbol "\" denotes the symmetric set difference.

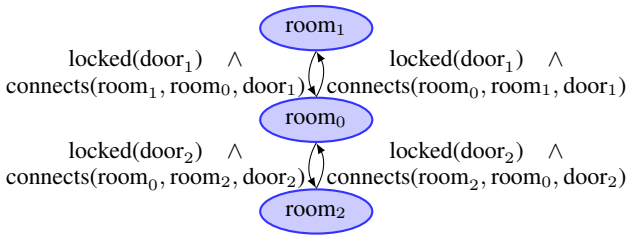


Figure 4: Domain transition graph for the SAS$^+$ variable *robot-pos* from the *Keys* domain.

Definition 8. *(Relevant domain) The relevant domain of a variable is the set of values necessary to achieve the goal.*

Definition 9. *(Variable contributing to the goal) A variable v contributes to the goal if adding or deleting an assignment to v from a state can make the goal true. Formally, v [3] contributes to $s^\star$ iff there exists a state s with $s \leq s^\star$ such that $s \cup \{v := x\}$ for some value x (Göbelbecker et al. 2010).*

The relevant domain of a variable $v \in \mathcal{V}$, $dom_{rel}(v)$, can be calculated using a fix-point iteration (Göbelbecker et al. 2010):

- If $v := x$ contributes to the goal, then $x \in dom_{rel}(v)$;
- For each variable v' in $dependentsOf(v)$, $dom_{rel}(v')$ contains the subset of $dom(v')$ which is (potentially) required to reach any element of $dom_{rel}(v)$.

Creating Fictitious Actions

The *Giving-Excuses* method tries to make the planning task solvable by two distinct ways:

- introducing actions that add new objects in a planning problem (*add-object* actions);
- introducing actions that change a variable value to x such that $x \in dom_{rel}(v)$ (*change-value* actions).

We call both *change-value* and *add-object* actions as *fictitious* actions, once they are not domain actions (i.e., actions specified in the planning domain).

Two variables are introduced: *started* and *unused*. The variable *start* is responsible for distinguishing domain actions from *fictitious* actions. This variable is *false* in the initial state and all the *fictitious* actions have $\neg$*started* as precondition and all the regular actions have *started* in their effects. The variable *unused* is responsible for controlling the new objects introduced. This variable is *true* in the initial state to indicate that the object was not yet used and each action have $\neg$unused in its preconditions (including the *change-value* actions).

Note that *change-value* action is responsible for modifying the value of a single variable. In order to find what are the *change-value* actions that should be created, we have to generate the relevant domain for each variable by traversing the causal graph, starting with the goal symbols. Then, we check in the *domain transition graph* if some relevant value is not being reached. For each variable v, every element $x \in$

[3]We are using the symbol := to indicate a variable assignment.

Action	Precondition	Effect
add(obj)	unused(obj) $\wedge$ $\neg$started	$\neg$unused
set_x^v	$\neg$started $\wedge$ $v(p_1, \cdots, p_n) = x$ $\bigwedge_{i=1}^{n} \neg\text{unused}(p_i)$	$v(p_1, \cdots, p_n) = x'$

Table 1: Description of the fictitious actions (Göbelbecker et al. 2010).

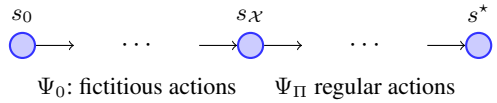


Figure 5: A plan solution for the plan task μ composed of fictitious actions and regular actions.

dom(v) from which $dom_{rel(v)}$ is not reachable is stored in a set called *changes*(v). Thus, for each variable v and $x \in$ *changes*(v), a new *change-value* action set_x^v is introduced. In Table 1 we have the description of the fictitious actions.

Without loss of generality we can add a certain fix number of new objects (add the corresponding add-objects actions) before starting the process of creating the set of *change-value* actions, in order to also consider variables involving the extra objects.

Selecting Fictitious Actions for Giving Excuses

To finally find excuses, a SAS$^+$ planner (e.g., fast downward planner) is used to solve the original problem. This planner considers s_0 as the initial state and s^* as the goal state and applies only fictitious actions in the prefix of the plan until achieving a new state $s_{\mathcal{X}}$ (called excuse state) from which it can reach the goal state s^* (only using domain actions).

Thus, the variable *started* will divide the plan into two parts:

- Ψ_0, before *started* become *true*, is a plan composed only by *fictitious* actions and;
- Ψ_Π, after *started* become *true*, is a plan composed only by *domain* actions.

Giving-Excuses Method with a Propositional Language

The *Giving-Excuses* method can be formally defined based on a propositional planning task description (instead of using SAS$^+$ language). This new approach relies on the ideas of symbolic model checking (Clarke et al. 1996).

Therefore, we start by describing the planning task described in a relational STRIPS-like language (Fikes and Nilsson 1972), such as PDDL (with STRIPS requirement) to induce a correspondent *propositional planning task*, i.e., involving a set of propositional state variables and propositional actions.

Let $\mathcal{O}$ be the set of PDDL operators of a given domain (with typed variables). A PDDL planning task can be written as in Figure 7.

```
(define (problem ⟨problem name⟩)
      (:domain ⟨domain name⟩)
      ⟨PDDL code for objects⟩
      ⟨PDDL code for initial state⟩
      ⟨PDDL code for goal specification⟩)
```

Figure 6: A PDDL problem (task) file scheme (Helmert).

add-object-object$_{\text{number}}$
pre: unused-object$_{\text{number}}$ $\wedge$ $\neg$started
eff: = $\neg$unused-object$_{\text{number}}$

Figure 7: The description of the propositional add-object action.

A propositional planning can be induced from a PDDL planning task, according Definition 10.

Definition 10. *(Propositional Planning Task) A propositional planning task is a tuple* $\Pi = \langle \mathcal{V}, \mathbb{A}, s_0, s^* \rangle$ *where:*

- $\mathcal{V}$ *is the set of propositional variables;*
- $\mathbb{A}$ *is the set of actions, with pre-conditions and effects.*
- s_0 *is the description of the initial state and;*
- s^* *is the goal specification.*

Creating propositional *add-object* (fictitious) actions

As in the *Giving-Excuses* method, we can start the whole process by adding new extra objects (candidates for excuses). We then add the fictitious actions of type *add-object-obj*$_{number}$, directly in the PDDL code for objects (Figure 7) with the following description:

- STEP 1: Add a set of extra objects X to the list of objects in the PDDL code for objects (Figure 7);
- STEP 2: To each included extra object, add to the initial state a fact *unused-obj*$_{number}$;
- STEP 3: Add $\neg$*unused-obj*$_{number}$ to each action involving variables of the same type of the obj$_{\text{number}}$.

Creating propositional *change-value* (fictitious) actions

This section formalize most of the work done by the *Giving-Excuses* method. As we will show, we generate the *change-value* set of actions without explicitly constructing the causal graph and the domain transition graphs. I.e. they are used implicitly in the logical operators described bellow.

We define a new set of propositional variables $\mathbb{P} = \mathcal{V} \cup \{v' : v \in \mathcal{V}\}$ where v represents the variable before the execution of an action and v' represents the variable after the action execution. This means that, in our representation, the variables v will occur in the action preconditions and variables v', in the action effects.

$\varphi_a \equiv$ robot-pos-room$_0$ $\wedge$ robot-pos-room$_2$' $\wedge$ $\neg$locked-door$_2$ $\wedge$ connects-room$_0$-room$_2$-door$_2$

Figure 8: An example of a propositional action a (move-room$_0$-room$_2$-door$_2$) from the *Keys* domain. Primed variables indicate the variable value in the next state.

Definition 11. *(Propositional Actions) An action $a \in \mathbb{A}$ can be represented as a boolean formula φ_a*

$$\varphi_a \equiv \bigwedge_{v \in pre(a)} v \quad \wedge \quad \bigwedge_{v' \in eff(a)} v'$$

where $pre(a)$ is the precondition of the action a and $eff(a)$, its effect.

Figure 8 shows a propositional action move-room$_0$-room$_2$-door$_2$ that moves the robot from room$_0$ to room$_2$ through the door$_2$. This action can be written as a boolean formula where the preconditions are represented by the variables v and the effects, by the variables v'.

Definition 12. *(Constraining the variable values in a boolean formula) Let φ be a boolean formula and v a variable. We denote $\varphi[v \backslash \bot]$ (or $\varphi[v \backslash \top]$) as a boolean formula obtained by replacing all occurrences of v in φ by $\bot$ (or $\top$).*

For example, if we restrict the value of variable robot-pos-room$_2$' to *true* ($\top$) in the boolean formula φ of Figure 8, we have: φ_a[robot-pos-room$'_2 \backslash \top$] = robot-pos-room$_0$ $\wedge$ $\neg$locked-door$_2$ $\wedge$ connects-room$_0$-room$_2$-door$_2$.

Definition 13. *(Relaxing the constraint for a variable in a boolean formula) We denote $\exists v.\varphi_a$ as the formula φ_a with the constraint on the value v relaxed, i.e., $\exists v.\varphi_a = \varphi_a[v \backslash \bot] \vee \varphi_a[v \backslash \top]$.*

For example, if we relax the restriction of the variable locked-door$_2$ in the boolean formula φ_a of Figure 8, we have: $\exists$ locked-door$_2$. φ_a = robot-pos-room$_0$ $\wedge$ robot-pos-room$_2$' $\wedge$ connects-room$_0$-room$_2$-door$_2$.

Now we have to extract the *relevant domain* (Definition 14) for the variables on which the sub-goal $g_i = x_i$ depends, i.e. *dependentsOf*(v) (Definition 6). As we will show, by using the boolean formula representation for actions, and the constraining and relaxing operations (Definitions 12 and 13) we can obtain the relevant domain without having to generate and use the causal and the domain transition graphs (Definition 6 and 7, respectively).

Let φ_a be a boolean formula that represents the action $a \in \mathbb{A}$ and let $v := x$ be an assignment for the variable $v \in \mathcal{V}$, with $x \in \{\top, \bot\}$. Note that if $v := x$ occurs in the a effects, the variable v' does not occur in the constrained formula, i.e., $\varphi_a[v' \backslash x]$. Thus, to know which variables are involved in the preconditions and effects of a and their relevant values, it is necessary also to relax the constraint of v in the formula φ_a.

Definition 14. *(Relevant domain) Let a subgoal $g_i := x_i$ be one of the effects of an action $a \in \mathbb{A}$. For each assignment $v := x$ that occurs in the preconditions and effects of a (excepting the variable g_i), we have to define:*

$$dom_{rel}(v) = \bigcup_{a \in \mathbb{A}} dom_{rel}(v)_a$$

where

$$dom_{rel}(v)_a = \{x\}$$

is the relevant domain of v according to an action a and is composed by the x of the resulting formula:

$$\exists g_i.\varphi_a[g'_i \backslash x_i] = \bigwedge (v := x)$$

In a next step, we recursively repeat the process of constructing the relevant domain for each subgoal $v := x$ until it is not changed anymore (fix-point iteration).

For example, giving the goal *robot-pos-room$_2$* we can extract the relevant domain for each variable v on which *robot-pos-room$_2$* depends according to the action move-room$_0$-room$_2$ (formula φ in the Figure 8). The relevant values for the variables robot-pos-room$_0$, locked-door$_2$ and connects-room$_0$-room$_2$-door$_2$ (according the action move-room$_0$-room$_2$) are respectively *true*, *false*, *true*.

Knowing the relevant domain for each variable, we can now verify if all relevant values can be reached using the regular actions of the planning domain (which is done in the previous section using the *domain transition graphs*). Since now all variables v are propositions if, e.g., the relevant domain of v is equal to $\top$, then we only have to verify if some action modifies its value from $\bot$ to $\top$. If there is no action doing such modification, it is necessary to create a *change-value* action for this change.

Let x be the relevant value of v. If an action $a \in \mathbb{A}$ has $v := \bar{x}$ in its preconditions[4] and $v' := x$ in its effects, then the formula $\varphi_a[v \backslash \bar{x}][v' \backslash x]$ has two variables less than φ_a. This restriction will be done for each variable v, value $x \in dom_{rel}(v)$ and action $a \in \mathbb{A}$ and its result is verified. If there is some relevant value of the variable v that is not reachable, then we create the *change-value* action set^v_x such that *pre*$(set^v_x) = \bar{x}$ and *eff*$(set^v_x) = x$. Notice that the action set^v_x has yet to be modified in order to respect domain constraints, e.g., "if the robot location changes to room$_1$ it should no longer be in any other room").

The set of set^v_x actions generated as we have just described, corresponds to the set of *change-value* actions generated by the SAS$^+$ *Giving-Excuses* method. By calling a propositional planner, we can finally return the same set of excuses $\mathcal{X} = \langle \mathcal{C}_\mathcal{X}, s_\mathcal{X} \rangle$.

Implementation and Empirical Results

We have developed a symbolic implementation of this propositional *Giving-Excuses* method, using a BDD library called JAVABDD (Whaley 2010). Basically, two BDD operations are used in the algorithms: *restrict*(x, v, B) that applies the constraint on the variable v to a boolean value x in the BDD B and; (ii) *exist*(v, B) that performs the relaxation of the constraint of v on the BDD B.

As the results in Table 2 shows, the generation of the fictitious actions has consumed reasonable time to generate excuses to the problems of the Logistics domain (IPC'00).

[4]The notation $\bar{x}$ means the inverse of the value x, i.e., if $x := \top$ then $\bar{x} = \bot$ and if $x := \bot$ then $\bar{x} := \top$.

problem	propositions	fictitious actions	time
logistics-04	180	212	0.275s
logistics-06	180	212	0.409s
logistics-08	207	202	1.082s
logistics-10	328	329	2.135s
logistics-12	328	329	1.969s
logistics-14	526	520	5.782s

Table 2: Results for creating fictitious actions on Logistics domain. All experiments were conducted on a 2.0 GHz Intel core i7 processor.

Planning as Model Checking

Model Checking consists in solving the problem

$$(\mathcal{M}, s_0) \stackrel{?}{\models} \varphi$$

where $\mathcal{M}$ is a formal model of a system, represented by a Kripke structure (Kripke 1963), s_0 is an initial state and φ is a CTL (Clarke and Emerson 1982) (Computational Tree Logic) property to be verified in this system. Essentially, a model checker is an algorithm that receives $(\mathcal{M}, s_0, \varphi)$ as input and systematically visits the states of the model $\mathcal{M}$, in order to verify if the property φ holds from the initial state s_0. If $(\mathcal{M}, s_0)$ satisfies property φ, the model checker returns success; otherwise, it returns a counter-example (e.g., a state in the model $\mathcal{M}$ where the property φ is violated).

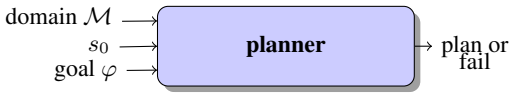


Figure 9: Planning as model checking approach.

In the planning context, the formal model can represent the planning domain model and the property φ, the goal formula. In this case, symbolic model checking can be used to solve the problem:

$$(\mathcal{M}, s_0) \models \neg\varphi$$

and the returned counter-example is a plan Ψ for φ. This planning technique is known as *planning as model checking* (Figure 9) (Cimatti et al. 2003). As a logical consequence we have that the plan satisfies φ:

$$(\Psi, s_0) \models \varphi$$

An important model checking technique is the computation of the pre-image to a set of states, as in Definition 15. We claim that the same technique can be used to generate a new initial state for an unsolvable planning problem, without having to generate fictitious actions, neither calling a planner.

Definition 15. *(Pre-image) Given a Kripke structure $\mathcal{M} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$, the pre-image of a set of states $Y \subseteq S$ is given by:*

$$\textit{pre-image}(Y) = \{s : s \in \mathcal{S}, a \in \mathbb{A} \textit{ and } \mathcal{T}(s,a) \cap Y \neq \emptyset\}$$

where $\mathcal{S}$ and $\mathcal{T}$ sets are, respectively, the states and transitions of the Kripke structure.

Minimal possible changes in the initial state based on model checking

We can use model checking techniques as an alternative way to propose changes in the initial state of a given problem for which we do not find a plan. For that, we apply the *pre-image* (Definition 15) (with fixpoint iteration) in order to find out the set of reachable states (for an unsolvable planning task, this set obviously does not include the initial state). The states with minimal difference w.r.t the original initial state corresponds to *acceptable excuses* states.

Algorithm 1 describes this process. Initially, we have a planning domain model $\mathcal{M} = \langle S, L, T \rangle$, an initial state $s_0 \in \mathcal{S}$ and a formula φ specifying the goal. The algorithm starts computing the pre-image of the set of goal states (i.e., the states satisfying the formula φ, which can be returned by a model checking algorithm, as the SAT function (Huth and Ryan 2004)). This is done until a fix-point is achieved (in this case, $X = Y$). If the problem has a solution, it is possible to reach the initial state s_0 (line 6). Otherwise, the set of reachable states Y is returned (line 7). Furthermore, the set of states $Z \subseteq Y$ that has a minimal difference w.r.t s_0 are *acceptable excuses* states.

Algorithm 1 GIVINGEXCUSES$(\mathcal{M}, s_0, \varphi)$

1: $X \leftarrow \mathcal{S}$
2: $Y \leftarrow$ SAT(φ)
3: **while** $X \neq Y$ **do**
4: $\quad X \leftarrow Y$
5: $\quad Y \leftarrow Y \cup$ preImage(Y)
6: $\quad$ **if** $s_0 \in Y$ **then**
7: $\quad\quad$ **return** "solvable problem"
8: **return** Y

In order to find the *good excuses* states, we have to select (for each path) a farthest *acceptable excuse* state from the goal. In Figure 10, for example, if the states s_1 and s_3 are both acceptable states, the algorithm has to return only s_1 as a *good excuse* state.

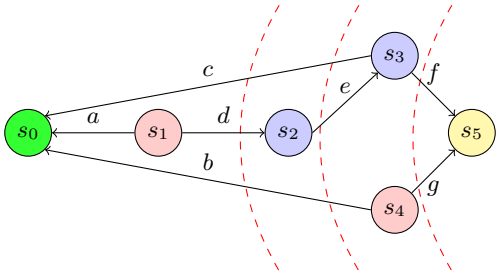


Figure 10: Starting from the goal state s_5, it is not possible to reach the initial state s_0. The GIVINGEXCUSES algorithm returns a set of states, where the minimal ones w.r.t s_0 are *acceptable excuses* states.

Modifying domain actions

The algorithm GIVINGEXCUSE can be modified to return not only the reachable states but also a set of pairs *state-action*.

Because the task is unsolvable there is no way (using the domain actions) to connect s_0 to some state in a path containing $s_{\mathcal{X}}$. An alternative way is to modify the planning domain actions. For this, we can use an approach called *Planning Domain Update* (Menezes, Pereira, and Barros 2010), which is able to relax or constrain preconditions and effects of the domain actions. By doing so, it is possible to connect s_0 to some path containing $s_{\mathcal{X}}$. Planning Domain Update is useful when we assume that the initial state is correctly specified and the domain actions are not.

Conclusion

In this paper, we have discussed the planning task validation problem. We summarized the *Giving-Excuses* method based on SAS$^+$ language, which is able to find excuses for an unsolvable planning task (i.e., generating a new initial state). This method constructs the fictitious actions based on the causal graph and the domain transition graphs for the planning domain. Our contributions are both theoretical and practical. On the theoretical side, we have formalized the *Giving-Excuses* method using propositional logic. On the practical side, we have implemented this formalization using Binary Decision Diagrams, resulting in a system that is capable of constructing the fictitious actions for propositional planning tasks. As in the *Giving-Excuses* method, these fictitious actions can be used by a planner in order to find a new initial state or a new set of objects. We also discussed how to generate the excuse states based on model checking techniques. By doing so, it is not necessary to generate fictitious actions neither call a planner. Finally, we briefly showed that when the initial state is correctly specified, we can use model update techniques to change the domain actions specification, in order to connect the initial state to some good excuse state.

Acknowledgements

This research is supported by FAPESP under grant no. 2010/10845-0.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11(4):625–655.

Brenner, M., and Nebel, B. 2009. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems* 19(3):297–331.

Bryant, R. E. 1986. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35(8):677–691.

Bryant, R. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)* 24(3):293–318.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Clarke, E. M., and Emerson, E. A. 1982. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *LNCS - Logic of Programs, Workshop*, volume 131, 52–71. London: Springer-Verlag.

Clarke, E.; McMillan, K.; Campos, S.; and Hartonas-Garmhausen, V. 1996. Symbolic model checking. In *Computer Aided Verification*, 419–422. Springer.

Edelkamp, S., and Mehler, T. 2005. Knowledge acquisition and knowledge engineering in the ModPlan workbench. *International Competition on Knowledge Engineering for Planning and Scheduling* 26–33.

Fikes, R., and Nilsson, N. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3-4):189–208.

Fox, M.; Gerevini, A.; Long, D.; and Serina, I. 2006. Plan stability: Replanning versus plan repair. In *Proc. ICAPS*, 212–221.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *Proc. ICAPS*, 81–88.

Helmert, M. An introduction to pddl. *16th AI Planning*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26(1):191–246.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proc. ICTAI*, 294–301. IEEE.

Huth, M., and Ryan, M. 2004. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge Univ Pr.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European conference on Artificial intelligence*, 359–363.

Khatib, L.; Muscettola, N.; and Havelund, K. 2001. Verification of plan models using UPPAAL. *Formal Approaches to Agent-Based Systems* 114–122.

Kripke, S. 1963. Semantical Considerations on Modal Logic. *J. Acta Philosophica Fennica* 16.

Lewis, D. 1973. Counterfactuals and comparative possibility. *Journal of Philosophical Logic* 2(4):418–446.

Menezes, M. V., and Barros, L. N. 2011. Model update for automated planning. In *AAAI/SIGART Doc. Consortium*.

Menezes, M. V.; Pereira, S. L.; and Barros, L. N. 2010. Model updating in action. *Workshop on Knowledge Engineering for Planning and Scheduling (ICAPS)*.

Menezes, M. V.; Pereira, S. L.; and Barros, L. N. 2011. System design modification with actions. *LNAI–Advances in Artificial Intelligence–SBIA 2010* 31–40.

Penix, J.; Pecheur, C.; and Havelund, K. 1998. Using model checking to validate AI planner domain models. In *the Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*. Citeseer.

Pereira, S. L., and Barros, L. N. 2008. A logic-based agent that plans for extended reachability goals. *Jornal of Autonomous Agents and Multi-Agent Systems* 16:327–344.

Van Der Krogt, R., and De Weerdt, M. 2005. Plan repair as an extension of planning. In *Proc. of the Int. Conf. on Automated Planning and Scheduling*.

Whaley, J. 2010. JavaBDD-java binary decision diagram library.

Yoon, S.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, volume 7, 352–359.

Zhang, Y., and Ding, Y. 2008. CTL Model Update for System Modifications. *Journal of Artificial Intelligence Research* 31:113–155.

EmergencyGrid – Planning in Convergence Environments

Natasha C. Queiroz Lino, Clauirton de A. Siebra and Manoel Amaro
Center of Informatics, Federal University of Paraíba
[natasha,clauirton]@ci.ufpb.br, manoel.amaro@lavid.ufpb.br

Austin Tate
Artificial Intelligence Applications Institute, School of Informatics, University of Edinburgh
a.tate@ed.ac.uk

Abstract

Government agencies are often responsible for event handling, planning, coordination, and status reporting during emergency response in natural disaster events such as floods, tsunamis and earthquakes. Across such a range of emergency response scenarios, there is a common set of requirements that distributed intelligent computer systems generally address. To support the implementation of these requirements, some researchers are proposing the creation of grids, where final interface and processing nodes perform joint work supported by a network infrastructure. The aim of this project is to extend the concepts of emergency response grids, using a convergence scenario between web and other computational platforms. Our initial work focuses on the Interactive Digital TV platform, where we intend to transform individual TV devices into active final nodes, using a hierarchical planning structure. We describe the architecture of this approach and an initial prototype specification that is being developed to validate some concepts and illustrate the advantages of this convergence planning environment.

Introduction

We have seen, in recent decades, a steady increase in natural catastrophes resulting in loss of life and physical damage. The earthquake in Haiti (2010, over 300,000 victims) and tsunami in Japan (2011, over 20,000 victims) are examples of such events. In fact, weather related events are expected to increase in number and severity in the future, due to the impacts of climate changes.

Nowadays, modern technologies could effectively impact the ability to plan, coordinate and respond to such disasters. These technologies are related, for example, to emergency communications, earth observation and events monitoring. Interactive Digital TV (IDTV) is one of these technologies that are being used in the emergency domain as a way to warn people about emergency on time. The IDTV platform enables the configuration of an emergency warning broadcast system and the sending of alerts (earthquake, tsunami, etc.) to each device in the area covered. The alert signal uses some data space in one of the segments of the data stream, turns on all receivers, if turned off, and presents the alert information. An example of such alert is the *Earthquake Early Warning* (EEW), which was well-utilized with alert sound and emergency box superimposed on TV screen at time of the 2011 Tohoku earthquake and tsunami and many aftershocks in several days. In April 2011, the Chilean Subsecretary of Telecommunications also released a similar alert system.

With the planned coverage of 95% of the worldwide population with digital television, there are in fact opportunities for prompt deployment of public emergency warning systems via satellite or terrestrial TV network. This work proposes the extension of this IDTV use so that they can bring more advanced information rather than simple disaster warnings. In this new perspective, the idea is to consider each IDTV device as final nodes of a hierarchical planning and task support structure, so that all the components can be seen as an emergency grid. This grid should provide a convergence environment, integrating IDTV, Web and mobile phone platforms, so that they could change knowledge and services with each other.

To build such a grid, we have provided a semantic layer to the IDTV middleware, so that intelligent process support could be implemented on this layer, sharing knowledge and planning information via ontological descriptions. The central planning node is implemented using the *Knowledge as a Service* metaphor, so that planning resources can be accessed as a service.

The remainder of this work is organized as follows: the next section describes the main works about the use of intelligent systems in emergency response scenarios. Then, we discuss the general architecture of our approach and

technologies that we are using to create an emergency grid that involves the IDTV platform. After that, we illustrate the use of this architecture with the specification of an emergency response application. Finally, we comment on the main remarks and future research directions.

Intelligent Systems for Emergency Response

Recently, many projects and initiatives have been devoted to provide intelligent computational support for emergency management. The work of Wang *et al.* (2007), for example, proposes an algorithm for optimal emergency resource allocation scheme in order to solve collision problems among multiple disaster places and multiple resource suppliers. Also regarding resource manipulation, Liu (2004) proposes a *possibilistic* Petri net-based resource description language, and related matchmaking mechanism, to search for relevant resources over the Internet that can cooperate to prepare for and respond to environmental emergency situations. Specifications of multiagent architectures [Basak *et al.* 2011; Schoenharl and Madey 2006] and decision making support systems [Tufekci 1995; Hernandez and Serrano 2001] are also important contributions from the research community to disaster relief.

These and other works highlight two research directions: low level approaches (*e.g.* resource search and allocation algorithms) and more general approaches (*e.g.* architectures and decision support systems). A different kind of approach aims to integrate previous solutions, or systems from different parts, to create more sophisticated disaster response solutions [Fortier and Volk 2006]. In this context, we see the *Grid* metaphor as one of the main research trends.

A *Grid* is a geographically distributed computation platform that can enable users to access various computing resources via a uniform computational interface [Foster and Kesselman 1999]. In grid computing, a single big task is split into multiple smaller tasks which are further distributed to different computing machines. Upon completion of these smaller tasks, they are sent back to the primary machine which in return offers a single output. Examples of Grid applications in the emergency response domain are the e-Response [Potter *et al.* 2004] and FireGrid [Upadhyay *et al.* 2008] research programmes.

e-Response is a simulated scenario in which a distributed team of specialist scientists use CoAKTinG (Collaborative Advanced Knowledge Technologies in the Grid) [Buckingham Shum *et al.* 2002] tools to coordinate emergency environmental protection activities. The domain used was an oil spill in the Solent, a strait separating the Isle of Wight from the mainland of England. FireGrid is an integrated emergency response system for fires in built environments. The broad objective is to provide fire fighters with as much useful information as possible that enables them to make sound and informed judgments while tackling the fire. To achieve this goal, the system provides the continuous assessment of the state of the building, forecasting the likelihood of future events and conveying this information to the responders at the scene.

Setting a Convergence Planning Environment

While all works discussed in the previous section are targeted at providing support for emergency response teams, we take a different approach, whose aim is to support civilians in processes such as evacuations of unsafe areas. In a similar way that FireGrid intends to provide fire fighters with useful information to support their decisions, our approach intends to also provide useful information to civilians, so that they can save themselves. For that end, common domestic devices, such as TVs and mobiles phones, should be used. This paper, in particular, focuses on the IDTV platform. The next sections discuss the technologies that we are using to extend the use of intelligent resources to this platform, creating a convergence environment where planning activities and their outcomes can be better delivered to normal civilians.

General Architecture

Figure 1 shows a conceptual view of the system. The *Planning and control center* composes the main node of the grid and it accounts for providing the planning services. To that end, it is being implemented in accordance with the *Knowledge as a Service* (KaaS) [Beijun 2010] metaphor. When TV devices receive a broadcast that contains warnings about a disaster, they inform their users about this disaster using messages and sounds in the display. This is the normal procedure in current emergency warning systems. However, this message also asks users to press a button on their remote control to get instructions about disaster procedures and actions to be carried out. An example is discussed latter on in this paper.

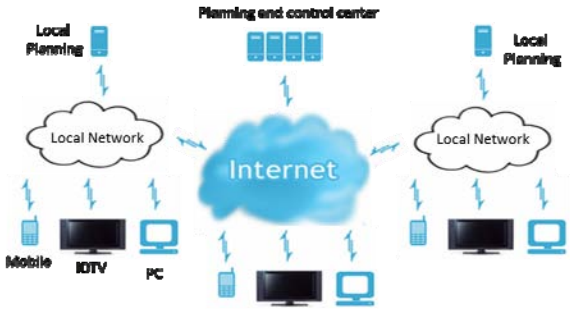


Figure1. Conceptual view of a convergence environment

Note that we may have local planning nodes to provide scalability to the system. In this case we can have three or more levels in the planning hierarchy. Several works present proposals about how to control and coordinate components in a hierarchical planning structure [Durfee and Montgomery 1991; Cox *et al*. 2005; Clement and Durfee 2001]. In our case, we are using extensions based on the I-X architecture [Tate 2000], which can be seen in [Siebra and Lino 2006]. However this discussion is out of the scope of this paper, so that we focus on the creation of the convergence environment and its extension to other platforms.

IDTV Architecture

To provide support to more advanced applications, we have created a *semantic* layer as part of the IDTV middleware. In fact, without this layer, the IDTV platform suffers from the same limitations as the World Wide Web. Current computational processes that run on the Web only account for leading the information transport, so that they do not have access to the meaning of the page content. The main reason is the form in which the information is structured, which is appropriate to the human user manipulation rather than computational processes. Thus, today we have a Web of documents rather than a Web of information, where computers can only provide limited assistance during the access and processing of information

The Semantic Web [Shadbolt *et al*. 2006] is the main W3C resultant technology for the problem discussed above. Its aim is to enable machines to understand the meaning of information on the Web. Some of its advantages are: sharing and reuse of data in different applications, automatic processing of data by computers, and semantic connections between data and the real world.

As we see, semantic representations are mainly important for systems integration and information sharing. Such features are the fundamental basis for a convergence environment. The Coalition Search and Rescue Task Support (CoSAR-TS) [Tate *et al*. 2006] is a good example of planning integration to other web services, supported by a semantic web environment. Emergency response operations by nature require the kind of rapid dynamic composition of available services making it a good use case for Semantic Web technologies.

IDTV Semantic Data Format

In the current IDTV standards, transmission of information, in a broadcast stream, is purely based on metadata definitions of tables and information services. The SI (Service Information) tables extend the PSI (Program Specific Information) tables, of the MPEG-2 standard, defining a set of structures that have descriptive data that transport specific IDTV information. Table 1 transcribes part of the MPEG-2 PSI/SI metadata table, which shows the fields 41, 42 and 43 related to the definition of an emergency alert.

The use of such tables facilitates the creation, processing, and rapid extraction of information. However, the SI tables are considered rigid metadata. Many services need more detailed information that cannot be satisfactorily defined within the SI tables. To that end, we have provided an ontological description to the IDTV operational data, so that external processes can understand the semantic meaning of their elements.

Table 1 - Part of the MPEG-2 PSI/SI metadata table

#	Metadata	Source	Description
…	…	…	…
41	state_area_code	NIT/PMT	Target state to emergence information transmission
42	microregion_area_cod	NIT/PMT	Target micro-region to emergence information transmission
43	signal_level	NIT/PMT	Specific emergency alert, which is defined by government organizations

In the proposed ontology, for example, we have the *EmergencyAlert* class. This class represents a signaling element that is transmitted by content providers to inform the population of a specific region about an imminent emergency situation. Another important element of this ontology is the *MMContent* class that represents a generic multimedia content entity and is the basis for all content construction that is used in the IDTV platform. The *EmergencyAlert* and *MMContent* are related by the *isEmergencyAlertTransmittedInto* property. This property indicates that a specific emergency alert is contained into a specific multimedia content during the IDTV transmission. Similarly the *hasLocationAlertFor* property relates the *EmergencyAlert* and *GeographicArea* classes. It indicates the scope of an emergency alert in terms of a geographic area.

Planning as a Service

In the proposed architecture, planning activities are mainly carried out in a server, rather than middleware. This approach is justified because such planning activities require a high processing power and data manipulation. This is a constraining factor, since current set-top-boxes do not have high processing power. In addition, another reason is that the middleware native operations have priority over computing resources usage. As a consequence, for instance, if the middleware needs more memory or processing power, it can demand computational

resources that are being used by an upper level application and all data can be lost. Thus, the demanding part of processes is being developed in accordance with the KaaS [Beijun 2010] paradigm, so that set-top-boxes only need to send and receive information from/to such services, carrying out simple parts of the whole planning process. Another motivation to allocate the whole demanding process in a server is the easier access from/to any other computational process and available data. For example, we can integrate services from other computational platforms, such as mobile and personal computers, and also compose new services using other available web services.

Two main advantages of the KaaS paradigm can be stressed. First, the models used by this paradigm are based on formal semantic representations, so that we do not have the same problems that are found in other web services. Second, the knowledge servers have the capacity of accessing data from different sources, instantiating their representations and generating knowledge to be delivered via intelligent process such as a distributed planning algorithm. Figure 2 shows a conceptual view of a service, according to the KaaS approach.

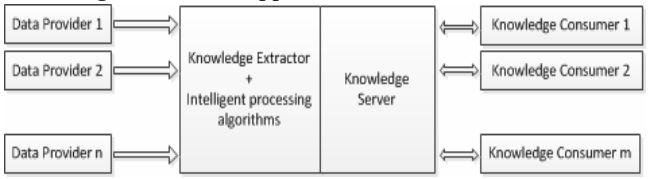


Figure 2. KaaS conceptual view [Xu and Zhang 2005].

According to this figure, the KaaS framework defines three logic components: (1) Data Providers, (2) Knowledge Server (Knowledge Extractor and Intelligent Processing algorithms) and (3) Knowledge Consumers. Considering our approach, Data Providers are sources of useful information that can assist the plan creation. For example, if the planning aim is to allocate tasks for emergency response teams, data providers could be represented by police stations, fire brigade centers and hospitals. The *Knowledge Server* runs a hierarchical multiagent planning algorithm, which is discussed in the next section. Finally, the *Knowledge Consumers* are represented by civilians, which can access emergency procedures via domestic devices, such as TVs and mobile phones.

The work of Paik *et al.* (2006) discusses some issues about the configuration of planning as a service and describes a framework for intelligent semantic web services that supports planning and scheduling aspects by a combined HTN planner and CSP (Constraint Satisfaction Problems) techniques. Note that the planning as a service approach is different from other approaches, which use planning mechanisms for the Web services composition problem [Traverso and Pistore 2004; Bo and Zheng 2009]. In the former case, planning is in fact the service, while this latter approach uses planning to compose the most diverse kinds of services.

Planning Aspects

The planning server is being specified in accordance with the I-X technology [Tate *et al.* 2006], which intends to provide a well-founded approach to allow humans and computer systems to cooperate in the creation or modification of some product, such as a plan. The use of I-X is justified because its planning representation is based on a formal ontology, called <I-N-C-A> (<Issues – Nodes – Constraints – Annotations>) [Tate 2003]. Thus, this ontology can be represented in the IDTV semantic layer as a domain ontology.

The main role of I-X planning agents is to provide actions to decompose higher level; more abstract activities until there are only executable activities. The important point in this discussion is to know that each planning step is implemented by an *activity handler*, which propagates the components through *constraint managers* to validate their constraints. Thus, all agents have a set of activity handlers that they use to refine or perform their activities. In a general way, the process follows these steps:

1. When an activity a is received, the agent's controller component selects a set H of activity handlers, which matches the description of a;
2. Each handler $h \in H$ uses one or more constraint managers to return its status (possible, impossible or not ready);
3. An optimal strategy, or an user, chooses one of the proposed handlers, committing to the performance of a;
4. During the execution, constraint managers are still monitoring the constraints of a, warning in case of problems, and maybe proposing continuations.

The role of constraint managers in this process is to maintain information about a plan while it is being generated and executed. The information can then be used to prune search where plans are found to be invalid as a result of propagating the constraints managed by these managers. The principal advantage of using constraint managers is their modularity. We can design managers to deal with specific types of constraints, such as the types discussed here (e.g., temporal, resource, commitment, etc.)

Together, the constraint managers form the model manager of the agent. Each constraint manager considers a set of specific constraints in a well-defined syntax, based on the support provided to a higher level of the planner where decisions are taken. However, they do not take any decision themselves. Rather, they are intended to maintain all the information about the constraints they are managing and to respond to questions being asked of them by the decision making level [Tate *et al.* 2006].

IDTV Emergence Response Application

This section details how this approach will be evaluated via a practical prototype that is in ongoing development. The prototype scenario represents part of Joao Pessoa (JP), the eastern-most city in Brazil. According to some scientists, there is a small chance that a mega-tsunami, originated from an earthquake close to Canary Islands, can reach the coast of JP (Figure 3). This region has a high population density, so that a simple emergency alert can create serious problems. For instance, the disordered use of the five coast evacuation routes may create big traffic jams.

Figure 3. Map of Joao Pessoa city coast.

In the proximity of a tsunami event, the broadcasters send warning messages (Figure 4, left hand side), which are described via metadata, to be displayed by IDTV devices. We intend that when users press the green remote control button, an instance of the *EmergencyAlert* class is created and sent to the planning server in the form of a request, together with parameters that describe the users of this device and support the planning process. At the moment, we are considering only two parameters: user´s address and locomotion type.

Figure 4. Examples of interfaces in IDTV platform.

When the server receives a request, it tries to allocate the best route from the user´s address to one of the safe areas, considering the traffic already allocated. The planner also returns the time that users must evacuate their homes. The clock carries out a count down until zero. At this moment, users must press the green button and evacuate their homes (Figure 4, right hand side). Obviously, this process is only valid to users whose locomotion way is defined as “car”. Otherwise (walk, bus, taxi, bike, etc.), a simple message is returned, asking an “as soon as possible” evacuation.

The first activity of the planning server is to acquire information, from *Data Providers* (Figure 2), about the event. In this application, important data is related to locations of safe areas and likely remainder time to disaster. After that, the allocation is carried out on demand. Sometimes we may have a route allocation that seems longer and non optimal. This is an effect of the on demand feature of this system. In order, we cannot have a pre-defined plan in advance because the planning system does not know how many civilians will be in the area at the moment of the alert broadcast.

Replanning activities are also limited in this scenario, since civilians are not monitored and they lose the communication channel after leaving their homes. This can create serious problems. For example, consider that one of the routes is blocked due to an accident. Consequently, other routes should be generated for the vehicles that are using the blocked route. This problem will only be considered after the integration of the mobile phone platform into this convergence scenario.

While the IDTV semantic representation and communication protocol between middleware and server is complete, we are still working on the planning service, mainly in the implementation of activity handlers. Three main concepts of the I-X architecture are appropriate for our implementation:

- Support for activity monitoring: we intend initially to only use the green button feedback (Figure 4, right hand side) as an indication that the plan is being followed. Future versions, using the mobile phone platform, will tend to use more advanced monitoring approaches;
- Support for *Standard Operating Procedures*: pre-planned set of activities, which can be used in specific situations, can be implemented as activity handlers;
- Modular implementation of activity handlers: at this moment we have only one type of handler that is *AlllocateRouteAndStartTime*. However we can have several versions (algorithms) of this implementation, each of them as a different activity handler.

We intend to use a simulator, such as Hermes [Xithalis 2008] to evaluate different versions of this handler. This application is a simple network simulator that allows us to design a network for a city and observe the level of service it can provide, i.e. number of vehicles and total trip time.

Conclusions and Research Directions

This work discusses a planning architecture where emergency response activities are provided via a server,

according to the KaaS paradigm. This paradigm enables, among other features, an appropriate semantic description to data that comes from different platforms. Our main aim is to use the KaaS metaphor as a form to enable convergence among different computational platforms, such as the IDTV, mobile phone and Web. Our initial focus was on IDTV platform, where a complete semantic model was defined for its data. However, future versions intend to consider the mobile phone platform, mainly as a way to extend re-planning strategies and monitoring abilities.

We are still implementing the planning server; however some important requirements have already identified. The principal question is how to implement an optimization planning mechanism that can use the evacuation waiting time to re-plan routes. This re-planning must be carried out in real time and have low impact on unaffected users.

References

Basak, S., Modanwal, N., and Mazumdar, B. 2011. Multi-Agent Based Disaster Management System: A Review. *International Journal of Computer Science & Technology*, 2(2): 343-348.

Beijun, S. 2010. A Design Framework for Public Knowledge Service Platform. *Proceedings of the International Workshop on Knowledge as a Service*, Xiamen, China.

Bo, Y., and Zheng, Q. 2009. Semantic Web service composition using Graphplan. *Proceedings of the 4th IEEE Conference on Industrial Electronics and Applications*, Xian, China, pp. 459-463.

Clement, B., and Durfee, E. 2001. Performance of Coordinating Concurrent Hierarchical Planning Agents Using Summary Information. *Lecture Notes in Computer Science*, Vol. 1986/2001, pp. 214-230.

Buckingham Shum, S., De Roure, D., Eisenstadt, M., Shadbolt, N. and Tate, A. 2002. CoAKTinG: Collaborative Advanced Knowledge Technologies in the Grid. *Proceedings of the Second Workshop on Advanced Collaborative Environments*, Eleventh IEEE Int. Symposium on High Performance Distributed Computing, Edinburgh, Scotland.

Cox, J., Durfee, E, and Bartold T. 2005. A distributed framework for solving the Multiagent Plan Coordination Problem. *Proceedings of the Fourth International Joint Conference on Autonomous agents and Multiagent Systems*, Utrecht, The Netherlands.

Durfee, E., and Montgomery, T. 1991. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6): 1363-1378.

Fortier, S., and Volk, J. 2006. Defining Requirements for ad hoc Coalition Systems during Disasters. *Proceedings of the IEEE International Conference on Computational Cybernetics.* Budapest, Hungary, pp. 1 – 6.

Foster, I., and Kesselman, C. 1999. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Hernandez, J., and Serrano, J. 2001. Knowledge-based models for emergency management systems. *Expert Systems with Applications*, 20(2):173–186.

Liu, K. 2004. Agent-based resource discovery architecture for environmental emergency management. *Journal Expert Systems with Applications*, 27(1):77-95.

Paik, I., Maruyama, D., and Huhns, M. 2006. A Framework for Intelligent Web Services: Combined HTN and CSP Approach. *Proceedings of the IEEE International Conference on Web Services*, pp. 959-962.

Potter, S., Tate, A., and Dalton, J. 2004. Collaborative Task Support and e-Response, *AISB Quarterly*, No. 115, Winter 2004.

Schoenharl, T., and Madey, G. 2006. WIPER: A Multi-Agent System for Emergency Response, *Proceedings of the 3rd International Conference on Information Systems for Crisis Response and Management*, Newark, NJ, USA.

Shadbolt, N., Berners-Lee, T., and Hall, W. 2006. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3): 96-101.

Siebra, C., and Lino, N. 2009. Aspects of Planning Support for Human-Agent Coalitions. *Journal of the Brazilian Computer Society*, 15(4):41-55, Rio de Janeiro, Brazil.

Tate, A. 2000. Intelligible AI Planning, in Research and Development in Intelligent Systems. *Proceedings of the Twentieth International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, Cambridge, UK, pp. 3-16.

Tate, A. 2003. <I-N-C-A>: a Shared Model for Mixed-initiative Synthesis Tasks. *Proceedings of the Workshop on Mixed-Initiative Intelligent Systems* at the International Joint Conference on Artificial Intelligence, Acapulco, Mexico, pp. 125-130.

Tate, A., Dalton, J., Bradshaw, J., and Uszok, A. 2006. Task Support: Intelligent Task Achieving Agents on the Semantic Web. *Air Force Research Laboratory Technical Report AFRL-IF-RS-TR-2006-91*.

Traverso, P., and Pistore, 2004. Automated Composition of Semantic Web Services into Executable Processes. *Lecture Notes in Computer Science*, Vol. 3298/2004, pp. 380-394.

Tufekci, S. 1995. An integrated emergency management decision support system for hurricane emergencies. *Safety Science*, 20(1):39–48.

Upadhyay, R., Pringle, G., Beckett, G., Potter, S., Han, L., Welch, S., Usmani, A., and Torero, J. 2008. An Architecture for an Integrated Fire Emergency Response System for the Built Environment. *Proceedings of the. 9th IAFSS International Symposium on Fire Safety Science*, Karlsruhe, Germany.

Wang, S., Wang, Y., and Sun, J. 2007. An Optimized Emergency Resources Allocation Algorithm for Large-Scale Public Emergency. *Proceedings of the International Conference on Machine Learning and Cybernetics*, Hong Kong, pp. 119-123.

Xithalis, C. 2008. Synchronous Control Method for Persona Rapid Transit Systems. *Proceedings of the 10th International Conference on Application of Advanced Technologies in Transportation*, Athens Greece.

Xu, S., and Zhang, W. 2005. Knowledge as a Service and Knowledge Breaching. *Proceedings of 2005 IEEE International Conference on Services Computing*, 1:87-94, Orlando, FL, USA.