ICAPS 2011

21th International Conference on Automated Planning and Scheduling

Proceedings of the System Demonstrations

Freiburg, Germany – 14 June 2011

Edited by Piergiorgio Bertoli and Minh Do

The 2011 ICAPS conference is sponsored by

Adventium Labs
Artificial Intelligence Journal (AIJ)
ATRiCS Advanced Traffic Solutions
David E. Smith
European Coordinating Committee for Artificial Intelligence (ECCAI)
European Office of Aerospace Research & Development
European Space Agency (ESA)
Florida Institute for Human & Machine Cognition (IHMC)
IBM Research
Institute for Computational Sustainability (ICS)
National ICT Australia Limited (NICTA)
National Science Foundation (NSF)
Robert Bosch GmbH
SICK AG
SIFT
TRACLabs Inc.
University of Freiburg, Faculty of Engineering

and held in cooperation with the Association for the Advancement of Artificial Intelligence

Twenty-First International Conference on Automated Planning and Scheduling

Proceedings of the System Demonstrations

Edited by
Piergiorgio Bertoli and Minh Do

Contents

Papers

Preface

The Demonstrations and Exhibits programme at ICAPS 2011 provides an opportunity for planning and scheduling researchers and practitioners to demonstrate their state-of-the-art implementations in action, allowing the community to experience the latest contributions while broadening the reach of novel methods in a relaxed social setting.

In 2011, the programme goes along the lines of well-established antecedents, while retaining some specific aspects. In particular, the session is co-located with the Doctoral Consortium event to exploit a synergy between the two events and improve the possibility for the audience to witness novel and exciting ideas from young students as well as from established researchers and research groups. Furthermore, the award for the Best Demonstration is chosen, in 2011, through a ballot.

The Demonstrations and Exhibits programme features twenty systems, ranging from deployed systems to research prototypes. Six of these systems were described in papers accepted to the main ICAPS conference; the remaining ones are either ad-hoc submissions to the Demo programme, or are related to works presented as part of ICAPS workshops. The twenty systems which are witnessed in the session, and in these proceedings, illustrate a wide variety of approaches, techniques and applications of planning and scheduling, spanning from tools that demonstrate the solutions to significant theoretical challenges to more practical and specific application-oriented tools aimed at real-world problems such as ship scheduling, training military forces, handling user agendas, or re-planning satellite constellations.

This proceedings of the 2011 Demonstrations and Exhibits programme contains abstracts and extended abstracts that describe the systems showcased. Systems described in papers presented in the main ICAPS conference are summarized here with abstracts; we refer to the conference proceedings for their full description. In addition to this proceedings, supplementary materials such as videos are found on the ICAPS website.

We would like to thank the ICAPS Chairs, Malte Helmert and Stefan Edelkamp, the Program Chairs, Fahiem Bacchus and Carmel Domshlak, the Workshop Chairs, Blai Bonet and Amedeo Cesta, and all workshop organizers, who supported us in identifying interesting entries to the Demos programme. We express our gratitude to all the people who helped raising the awareness of the event, to the Local Arrangement, headed by Gabriele Roger, and to the staff arranging and organizing the events in Freiburg.

We hope that the Demonstrations and Exhibits programme provides its attendees lively and fruitful discussions which may motivate and eventually bring promising approaches to scheduling and planning from the laboratory to the real world.

– Piergiorgio Bertoli and Minh Do
Demonstrations and Exhibits Co-chairs

FlowOpt: A Set of Tools for Modeling, Optimizing, Analyzing, and Visualizing Production Workflows

Roman Barták[1*], Martin Cully[2], Milan Jaška[1], Ladislav Novák[1], Vladimír Rovenský[1], Con Sheahan[2], Tomáš Skalický[1], Dang Thanh-Tung[2]

[1] Charles University, Faculty of Mathematics and Physics, Malostranské nám. 2/25, Praha, Czech Republic
[2] ManOPT Systems Ltd., National Technology Park, Limerick, Ireland
[*] bartak@ktiml.mff.cuni.cz (contact e-mail)

Abstract

FlowOpt is a collection of modules built on top of enterprise performance optimization system MAK€ with the goal to bring unique modeling, optimizing, and visualizing capabilities for production workflows. FlowOpt is based on the concept of nested temporal network with alternatives used to formally describe alternative processes in workflows. The system supports visual design of nested workflows that can be connected with available resources in the enterprise. It then generates production schedules optimizing the on-time-in-full performance criterion. The obtained schedules can be analyzed to discover and "repair" inefficiencies of the enterprise. Finally, the schedules are visualized in the form of Gantt chart where the user can do any schedule modification interactively including selection of an alternative process or change of resource allocation.

Introduction

Though there exists a vast amount of research in the area of scheduling there is still a large gap between practical problems and research results especially in the area of production optimization for small and medium enterprises. This gap is partly due to missing modeling and visualization tools that would allow easy transformation of real-life problems to optimization models and the results back to customers (Barták et al. 2010) and partly due to large distance of academic algorithms from the existing problems.

FlowOpt project is a student's software development project at Charles University in Prague (Czech Republic) that is used to demonstrate the recent research results in the areas of modeling and optimizing production workflows. The project was realized in close co-operation with an industrial partner, ManOPT Systems from Limerick, Ireland. This project is a unique opportunity as usually researchers and final customers are too far from each other to communicate directly the needs on one side and the possibilities on the other side. In the project we wanted to demonstrate the recent research advancements in the areas of formal problem modeling and solving, namely using Nested Precedence Networks with Alternatives (Barták and Čepek 2008) to describe workflows, in real-life industrial setting. FlowOpt is a collection of four modules built on top of commercial system MAK€ being developed by ManOPT Systems as a tool for enterprise performance optimization. In particular, FlowOpt consists of the Workflow Editor, the Scheduler, the Gantt Viewer, and the Schedule Analyzer. These modules deal with creating, managing, scheduling, optimizing, and analyzing manufacturing processes for (small to medium) enterprises. The general purpose is to provide a streamlined feature rich environment where the user could do all of the following in a simple, efficient and user-friendly way:

- Specify how a particular product is manufactured (i.e. define a workflow describing the manufacturing of a single product).
- Enter a work order from a customer – customers request certain quantities of products that the factory can manufacture, together with a desired deadline.
- Generate a schedule for the order – a schedule should be a complete description of what elementary activities should be performed, in what exact times should they run and what resources should they use (machines or people). Executing such a schedule should result in efficient fulfilling of the work order.
- Display the generated schedule in the form of a Gantt chart and modify it interactively.
- Analyze the generated schedule for possible opportunities of improvement.

FlowOpt Architecture

FlowOpt modules are implemented as plug-in modules on top of MAK€ system which provides all integration capabilities. MAK€ contains a database where information being passed between the modules is stored, it also adds some modeling tools, namely for description of bill of materials, resources, time windows, and custom orders and it does some data integration, namely connection of abstract workflows with particular data regarding custom orders. MAK€ is a fully-featured product that already provided workflow editor, scheduler, and Gantt viewer. The FlowOpt modules are used as alternatives there with the focus on specific structure of workflows – nested precedence networks with alternatives.

FlowOpt Workflow Editor

Workflow Editor is the first module the user is exposed to. This module allows users to create and modify workflows. Workflow is a basically a set of activities connected via temporal constraints. We adopted the idea of nested precedence network with alternatives (Barták and Čepek 2008) to specify the workflow structure. Briefly speaking the initial workflow consists of the single task to achieve some goal and the user specifies how to decompose the task into sub-tasks until real operations/activities are obtained. Three types of decompositions are supported (Figure 1), either the task is decomposed into a sequence of sub-tasks which forms a *serial decomposition* or the task is decomposed into a set of sub-tasks that can run in parallel – a *parallel decomposition* – or finally, the task is decomposed into a set of alternative sub-tasks such that exactly one sub-task will be processed to realize the top task – an *alternative decomposition*. The module supports fully interactive construction of workflows with both top-down method of constructing workflows by the decomposition operations and the bottom-up method where existing workflows/tasks are joined in a similar style. When the structure of the workflow is defined the user can fill the most inner tasks by real activities and can define required resources for these activities. All these operations can be realized in drag-and-drop style.

In addition to core nested structure that is exploited during scheduling it is possible to specify additional relations between the activities going beyond the nested structure. In particular, the user can specify additional *precedence relations* between any pair of tasks which means that if both tasks are selected in the solution then the specified ordering must hold. The system also supports *synchronization constraints* so it is possible to describe that two tasks start or end at the same time or that one task must start exactly when another task finishes. Currently we do not support more general temporal constraints as the above constraints seem enough for most production workflows. We however support special *logical relations* between the tasks, namely implication, equivalence, and mutex relations which allow the users to describe some causal relations between the tasks going beyond the nested (hierarchical) structure. Basically, these logical relations restrict which activities can/must appear together in the schedule, for example, mutex relations mean that both tasks/activities cannot appear together in the schedule. These logical relations are novel in scheduling though they are frequently used in planning. We believe they will further simplify modeling of real-life problems, but further evaluation from customers is necessary.

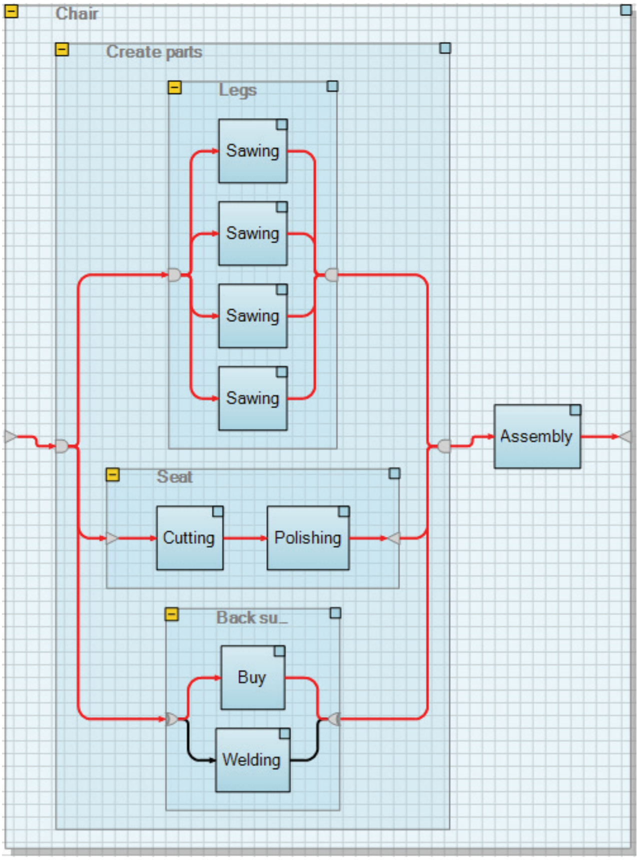
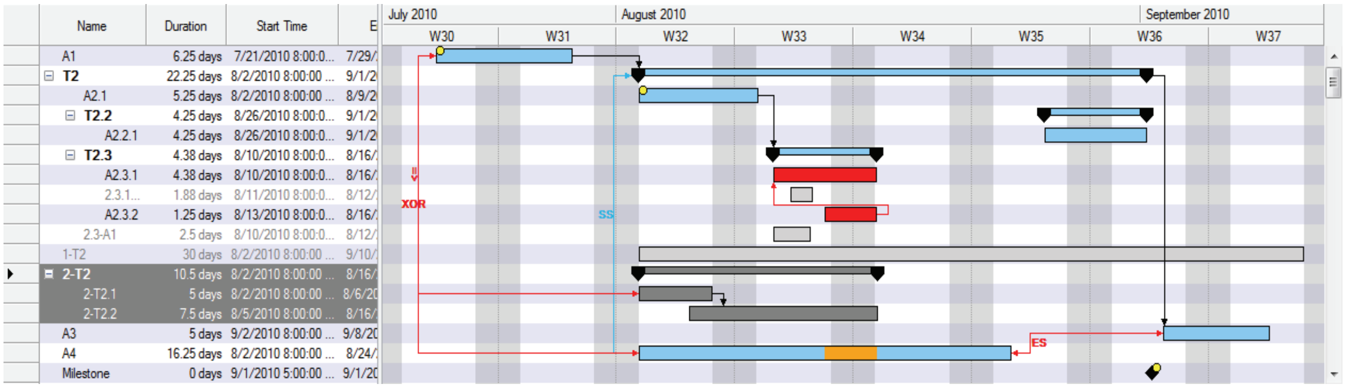


Figure 1: Visualization of nested workflow in FlowOpt Workflow Editor (from top to down there are parallel, serial, and alternative decompositions)

FlowOpt Scheduler

When the workflow is described, it must be filled by particular data from custom demands. This integration is done by the MAK€ tool using the techniques presented in (Barták et al. 2010). A complete description of the scheduling problem that contains activities organized in a nested structure, specification of required resources, and description of deadlines is passed to the FlowOpt Scheduler. We use the idea of optional activities so the



Name	Duration	Start Time	E
A1	6.25 days	7/21/2010 8:00:0...	7/29/
⊟ T2	22.25 days	8/2/2010 8:00:00 ...	9/1/2
A2.1	5.25 days	8/2/2010 8:00:00 ...	8/9/2
⊟ T2.2	4.25 days	8/26/2010 8:00:0...	9/1/2
A2.2.1	4.25 days	8/26/2010 8:00:0...	9/1/2
⊟ T2.3	4.38 days	8/10/2010 8:00:0...	8/16/
A2.3.1	4.38 days	8/10/2010 8:00:0...	8/16/
2.3.1...	1.88 days	8/11/2010 8:00:0...	8/12/
A2.3.2	1.25 days	8/13/2010 8:00:0...	8/16/
2.3-A1	2.5 days	8/10/2010 8:00:0...	8/12/
1-T2	30 days	8/2/2010 8:00:00 ...	9/10/
⊟ 2-T2	10.5 days	8/2/2010 8:00:00 ...	8/16/
2-T2.1	5 days	8/2/2010 8:00:00 ...	8/6/20
2-T2.2	7.5 days	8/5/2010 8:00:00 ...	8/16/
A3	5 days	9/2/2010 8:00:00 ...	9/8/20
A4	16.25 days	8/2/2010 8:00:00 ...	8/24/
Milestone	0 days	9/1/2010 5:00:00 ...	9/1/20

Figure 2: A task view in the FlowOpt Gantt Viewer. It displays the hierarchical structure of nested workflows, shows the additional constraints, and highlights the violated constraints.

system is doing some form of integrated planning and scheduling where activities used to satisfy the demands must be selected from the alternative processes and then allocated to time and resources. We use ILOG CP Optimizer to achieve this task as it already supports optional activities (Laborie and Rogerie 2008). To solve the scheduling problem we exploited only the existing constraints in CP Optimizer and its built-in search strategy, no special solving algorithm was developed. The result of scheduling is identification of selected activities that are allocated to precise time and required resources. The result from the Scheduler can go directly either to FlowOpt Analyzer or to FlowOpt Gantt Viewer.

FlowOpt Analyzer

FlowOpt Analyzer is probably the most innovative module in the FlowOpt system. It is responsible for analyzing an existing schedule and suggesting possible improvements of the enterprise such as buying a new machine. Note that these improvements are going beyond the analyzed schedule as they suggest how to change the enterprise rather than how to change the schedule only. The MAK€ system already includes schedule analyzer based on cumulative computing of certain key performance indicators such as utilization of resources or the number of late deliveries which are used to generate useful reports to the customers. The main difference from existing approaches in MAK€ is that the FlowOpt Analyzer does structural analysis of the schedule consisting of identifying critical activities (similarly to critical path detection) that cause delays of deliveries and finding a reason why these critical activities are itself delayed. Based on this analysis, the system automatically suggests possible modifications of the enterprise such as adding a new resource or exploiting overtime. These so called *improvement projects* are then evaluated by the scheduler simply by finding a new schedule after the modification and comparing quality of the new schedule with the quality of the original schedule. During the evaluation some inter-relationships between the improvement projects are also identified. All obtained information is then used in standard project portfolio optimization which will select a subset of most promising improvements.

FlowOpt Gantt Viewer

FlowOpt Gantt Viewer stands at the end of the modeling and solving process. As expected this module is responsible for visualizing schedules in the form of a Gantt chart. There are two major innovative ideas behind the Gantt Viewer. First, the module fully supports visualization of the nested structure of scheduled workflows including not-used alternatives. Second, the module allows interactive modification of schedules so the user can do fine tuning of the schedule. This interactive modification is similar to work (Barták and Skalický 2009) though automated schedule repair is not yet supported.

The Gantt Viewer allows both typical views of the schedule: a resource view that is useful to show occupation of individual resources (Figure 3) and task view showing the hierarchical structure of workflows together with time allocation (Figure 2). The task view provides visually more information as it shows not only the scheduled activities but also the non-selected alternatives and all these additional constraints between the activities as defined in the workflow editor. Moreover, the tool supports any modification of the schedule so users can change time allocation of tasks as well as resource allocation. This is realized by intuitive drag-and-drop operations. Users can also select alternative tasks if they are not happy with the choice of the automated scheduler. In such a case, the tool removes the previously selected alternative from the resources. Some other features are supported such as banning the resource so it cannot be used for activities or pinning the activity so it remains at the specified position. After manual modifications, the system highlights possible violated constraints though it does not repair them yet.

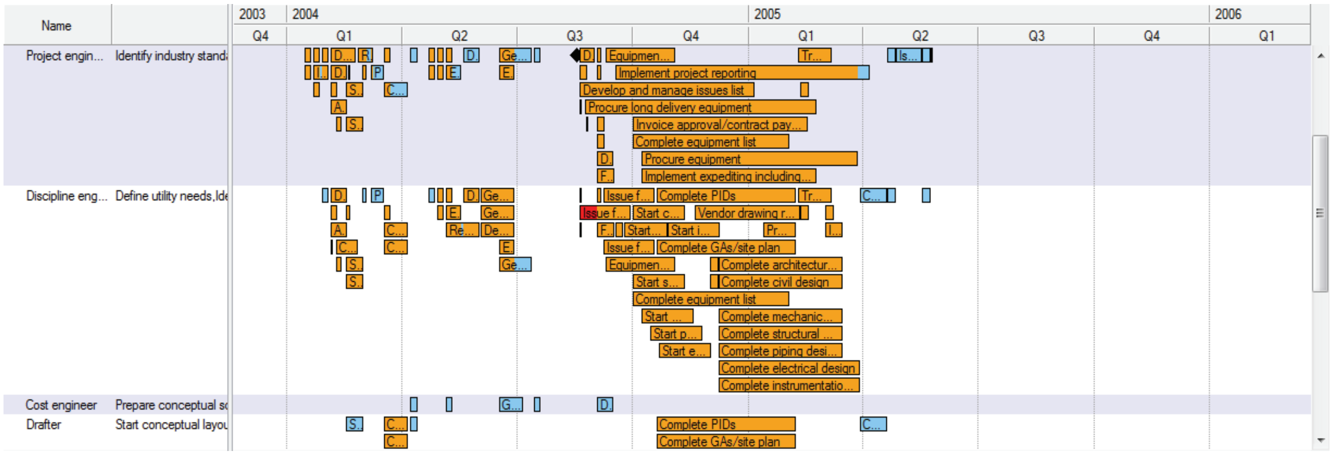


Figure 3: A resource view in the FlowOpt Gantt Viewer. It displays allocation of activities to individual resources.

Conclusions and Future Development

The MAK€ application has had a total of 15 years of use in five actual differing production facilities for example to schedule production of pistons or wooden doors. It uses a scheduling algorithm based on preferred routes specified by the users and some experiments were done with generic scheduling engine based on the constraint-based technology. The goal of the FlowOpt project is to test novel modeling techniques such as the nested structure of workflows and novel solving techniques such as using ILOG CP Optimizer. FlowOpt is not yet operational.

MAK€ with its FlowOpt extension is under continuous development driven by both customer requirements and novel research results. Three major extensions are planned for near future.

The Workflow Editor provides unique capabilities for hierarchical description of workflows with additional temporal and logical constraints. These additional constraints may introduce infeasibilities to the workflow, for example when the user wants to synchronize some activities, which may be impossible due to other temporal constraints. We are working on developing automated validation of workflows that can discover such infeasibilities and suggests the user how to remove them.

The Gantt Viewer on the other side of the modeling and solving process allows users to visualize and modify the obtained schedules. Any modification is allowed which may introduce conflicts to the schedule, for example some activity is manually allocated to a resource where the available capacity is exceeded. Currently, these conflicts such as violated constraints are highlighted to the user. The next version of the Gantt Viewer will support (semi-) automated schedule repair based on introducing additional changes to the schedule that will repair the conflicts while minimizing the number of changes and keeping the schedule quality (Barták and Skalický 2009).

The most novel part of FlowOpt is Schedule Analyzer that explores an existing schedule, detects possible inefficiencies, and suggests how to improve the quality of schedules by modifying the enterprise. We plan to extend this module to detect more inefficiencies and to suggest a wider spectrum of so called improvement projects. Also we assume a deeper integration with the Optimizer where the Analyzer can provide additional information for generating new schedules such as estimating where the bottlenecks appear.

Acknowledgements

The research is supported by the Czech Science Foundation under the contract P202/10/1188 and by EU Funding Scheme Research for the benefit of SMEs: FP7-SME-2007-1 under the project ValuePOLE (contract 222218).

References

Barták, R. and Čepek, O. 2008. Nested Precedence Networks with Alternatives: Recognition, Tractability, and Models. In D. Dochev, M. Pistore, P. Traverso (eds.): *Artificial Intelligence: Methodology, Systems, and Applications* (AIMSA 2008). LNAI 5253, Springer Verlag, pp. 235-246.

Barták, R., Little, J., Manzano, O., Sheahan, C. 2010. From Enterprise Models to Scheduling Models: Bridging the Gap. *Journal of Intelligent Manufacturing*, Volume 21, Number 1, pp. 121-132, Springer Verlag.

Barták, R. and Skalický, T. 2009. A local approach to automated correction of violated precedence and resource constraints in manually altered schedules. In *Proceedings of MISTA 2009: Fourth Multidisciplinary International Scheduling Conference: Theory and Application*s, Dublin, Ireland, pp. 507-517.

Laborie P. and Rogerie, J. 2008. Reasoning with Conditional Time-intervals. In *Proceedings of the Twenty-First International FLAIRS Conference*, AAAI Press, pp. 555-560.

Only *Hope* remains in the PANDORA's `.jar`

– Pervasive use of planning in a training environment –

G. Bernardi[†], **A. Cesta**[†], **L. Coraci**[†], **G. Cortellessa**[†], **R. De Benedictis**[†],
F. Mohier[‡], **J. Polutnik**[*] and **M. Vuk**[*]

[†] CNR, Consiglio Nazionale delle Ricerche, ISTC, Rome, Italy
[‡] BFC, Business Flow Consulting, Sceaux, France
[*] XLab Research, Ljubljana, Slovenia

Abstract

This paper shortly introduces features of a software system called PANDORA-BOX. [1] It shows a novel use of timeline-based planning as the core element in a dynamic training environment for crisis managers. A trainer is provided with a combination of planning and execution functionalities that allow him to maintain and adapt a "lesson plan" as the basis for the interaction between him and the involved trainees. The training session is based on the concept of *Scenario*, a set of events and connected possibilities that shape an abstract plan proposed to trainees through a timeline-based system. The PANDORA architecture provides a continuous reactive loop around trainees, and, additionally allows the trainer to directly intervene in the ongoing session giving him a complete, general and advanced view about the evolution of the *Scenario*.

Introduction

Goal of the PANDORA project[2] is to study how to support the training of crisis managers with innovative "ICT-like" technologies. In particular the project aims at creating a tool that corroborates with traditional training methods to generate the ability for trainees to react well to decision making under critical situations.

Why. When a catastrophic event occurs, it is often human behavior alone that determines the speed and efficacy of the crisis management efforts. Indeed, all too often, shortcomings in the response to the emergency do not stem from the ignorance of procedures but from difficulties resulting from the individual response to the challenge of operating in such a context, particularly when additional unexpected problems arise. Crisis management is of major importance in preventing emergency situations from turning into disasters. In recent years, poor management response to emergencies has often resulted in critical situations. In critical circumstances, there is a tremendous necessity of effective *leaders*. It is worth saying that the pressure of unexpected circumstances creates strong constraints on the abilities of leaders to take decisions. For example, given the severe time pressure imposed by the crisis, they have little time to acquire and process information effectively. As a consequence, they are required to assess information and making decisions under tremendous psychological stress and physical demands, often caused by the difficulty to operate in contexts where consistent losses as well as damages both to human lives and properties are occurring. Within this context training plays a crucial role in preparing crisis managers. Specifically, training for strategic decision making has to foster leaders' ability to anticipate possible consequences of bad decisions and to conceive creative solutions to problems. In this light experiential learning plays a crucial role.

What. The project is synthesizing a software environment able to support a lesson of few hours with a class of trainees that are exposed to a set of stimuli coming from an evolving crisis scenarios customized to the particular training needs. Key aspect in PANDORA is to create realistic responses to the decisions taken by trainees by reproducing believable situations, grounded realistic domain causality for those decisions to facilitate the development of a comprehensive range of decision making skills. Additionally, the idea underlying PANDORA is to take trainees behavioral features into account and plan training sessions tailored to individual differences and needs.

How. The starting idea for using planning within PANDORA was connected to the synthesis of a "Lesson plan", that is an organized set of lesson's items to be given to trainees over a time span according to a learning strategy. Additionally from the need of monitoring user status during lesson comes the idea of representing also the user's features as temporal items, hence inserting also these data in a uniform plan and using causal connections between different part of such plan to foster the continuous update of the plan. A natural technology for all this has been identified in the timeline-based planning, an approach to temporal planning which has been mostly applied to the solution of

[1] For those curious about the paper title we quote a short mythological description from the voice *Pandora's_box* on Wikipedia: "When Prometheus stole fire from heaven, Zeus took vengeance by presenting Pandora to Prometheus, Epimetheus' brother. With her, **Pandora had a box which she was not to open under any circumstance**. Impelled by her natural curiosity, Pandora opened the box-jar, and all evil contained escaped and spread over the earth. She hastened to close the lid, but **the whole contents of the jar had escaped, except for one thing** which lay at the bottom, **which was Hope**."

[2] http://www.pandoraproject.eu/

several space planning problems – e.g., (Muscettola 1994; Jonsson et al. 2000; Smith, Frank, and Jonsson 2000; Frank and Jonsson 2003; Cesta et al. 2011b; Chien et al. 2010). We have produced a first version of a comprehensive architecture, called the PANDORA-BOX, that fully demonstrates the feasibility of our approach. Central to the system is its original use of planning to model a quite rich domain. Specifically, planning is used (1) to compute diversified crisis scenarios corresponding to alternative training paths to foster creative decision-making, (2) to model and maintain trainees' behavioral patterns according to which training can be personalized, (3) to support mixed-initiative interaction between the trainer and the automated learning environment relying on a high level of abstraction for the internal representation. Here we describe PANDORA Year One Demo in a quite broad way. For a more detailed description the reader should refer to (Cesta et al. 2011a).

The PANDORA-BOX

Figure 1 describes the modules that currently compose the PANDORA-BOX system. At the more external level three are the main blocks of the current architecture:

1. the *Trainer Support Framework* which allows the trainer to keep control of the training session by biasing the learning content steps with an abstract plan called *Scenario*, dynamically adjusting the stimuli based on both his/her experience and observation of the different trainees' actions;
2. the *Trainee Clients* that according to a Client-Server communication allow distributed trainees to join a class and participate, also being able to receive both collective and individual stimuli during the class;
3. the *kernel* PANDORA *system*, identified by the dotted PANDORA-BOX in the figure, which is the main engine that generates the "lesson plan", animates it in an engaging way and adjusts it on a continuous bases to keep peace with both the evolution of the specific group of people under training and their individual performance.

In the rest of this paper we describe main aspects of these three blocks focusing in particular on the kernel because it is there that planning technology is more deeply used.

Planning the Training Class

The basic connection with planning relies on the idea of composing elements of the lesson through causal rules. In PANDORA-BOX lesson's content, e.g., different multi-media assets, are represented as elements of a temporal plan, hence the crisis plan is composed of different multi-media "messages" to trainees. Additionally, also all the background information, e.g., lesson strategy, trainee classification, evolution of crisis on-field resources, are represented as timelines to take advantage of both the uniform representation and the underlying technological functionalities. The combination of such information is useful to decide particular orchestration of messages.

One of the key points of our representation of the plan is the ability to adapt and update itself as a consequence of new information gathered from trainees during the ongoing lesson. Each action done by both the Trainer and the Trainees is figured as a trigger able to change the current running state of the backbone Scenario created by the trainer. As a consequence of this requirement, the system needs to activate a re-planning procedure in a continuous cycle in order to maintain the simulation consistent with taken decisions.

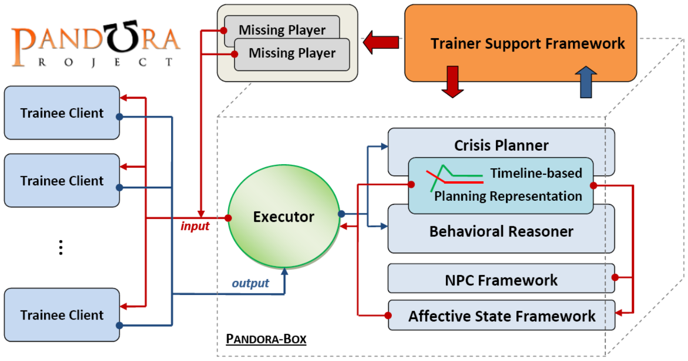


Figure 1: *The* PANDORA-BOX *general architecture*

Timelines-based modeling. As usual in timeline-based planning the basic indexing of domain knowledge is represented by timelines, that in generic terms, are functions of time over a finite domain (Muscettola 1994). A single timeline contains *a set of tokens* that we have called *"events"* here due to the association with visible effects on the whole played Scenario. Such events can have consequences in terms of casualties, injuries, involved resources, etc. or simply represent information sent to single trainees. From a technical point of view, an event is described through a predicate holding over a time interval and thus characterized by a start and an end time. According to this model, the domain of each timeline depends on the type of events that the same timeline is going to represent. Furthermore, events can be linked each other through "relations" in order to reduce allowed values for their constituting parameters and thus decreasing allowed system behaviors. Generally, relations can by represented by logical combination of linear constraints among event parameters and/or temporal points. We call the graph having events as nodes and relations as edges "Event Network" and we say that it is *consistent* iff it respects a set of consistency rules that we call "Causal Patterns". A causal pattern is a logic implication having a predicate signature as implicant and a logic combination of timeline values and relations as implicate. The semantic is that each node of the Event Network having the implicant pattern as signature requires the implicated pattern inside the Event Network.

The uses of plans. One aspect worth being observed in Figure 1 is how the PANDORA system creates loops around its human users. We can call the first one the *the-trainee-loop*: Trainees receive stimuli, their decisions are registered by the system and then reacted upon through plan adaptation, before loop continuation. The starting point for plan generation is the Trainer Support Framework because the trainer injects an initial Scenario (aka Abstract Plan) that acts as a connected set of goals when represented at the ground planning level. These set of goals triggers the basic planning activity of the Crisis Planner. The planner uses both the domain causal patterns and the timelines inputted by the Behavioral Reasoner with information on the single trainees to create a complete consistent plan at ground level that is ready for execution. The Behavioral Reasoner is the module responsible for both creating an initial user model of the trainees and maintaining it updated according to a continuous analysis of trainees decisions, and other data (Cortellessa et al. 2011). Two additional modules compose the PANDORA-BOX and are connected to an effective rendering of single events: the NPC Framework and the Affective State Framework. The first makes available additional virtual characters to be functionally used within the orchestrated events to influence trainees, the latter, at present, can be directly controlled by a timeline called `induced_stress` synthesized and updated by the Behavioral Reasoner to generate diversified multi-media effect to influence the engagement and the cognitive overload of the trainees. The Executor is the main responsible for the dispatching of events according to temporal order. It is also responsible for gathering decisions coming from trainees after selected stimuli and for forwarding them to the two modules that dynamically update the timeline plans (the Crisis Planner and the Behavioral Reasoner).

There is a second human-in-the-loop case that we can call *the-trainer-loop*: as shown in the figure this person observes what is happening in the class and can intervene on the trainees either *directly* through simulated characters (the Missing Players) and chat messages (not represented in figure), or *indirectly* by changing the Scenario and in so doing posting new goals at the ground planning level. In general the trainer has the possibility of just observing the lesson flow and annotate the abstract plan representation or more proactively taking part in the lesson or even interrupting it, giving direct explanations, and resuming the plan-based lesson. It is also worth saying that PANDORA provides another instrument that allows the Trainer *temporal navigation* through the lesson plan. A *Rewind* functionality allows to move the execution back in time providing two different behaviors:

- *default roll-back*, intended for debriefing purposes, that simply updates current simulation time t to desired target value keeping untouched actions taken by trainees;
- *heavy roll-back*, intended to revert to a crucial decision point at time t, removing each event representing trainees' choices at time $t' > t$, along with their consequences, in order to allow a different simulation course.

It is worth saying that the roll-back is a functionality of the Executor fully supported by the plan management machinery provided by the timelines. For the sake of space we have given a quite generic presentation of the PANDORA-BOX. One comment worth being done is that also in this experience we have noted, in agreement with (Pollack and Horty 1999), how in real applications as important as pure plan synthesis is the richness of services that can be developed around plan management.

The PANDORA interactive environment. We close this compact overview with a description of the functionalities realized to interface real users. Figure 2 depicts some of the interaction features in the current demonstrator. As direct consequence of the choices in the architecture, the system distinguishes between two types of interaction:

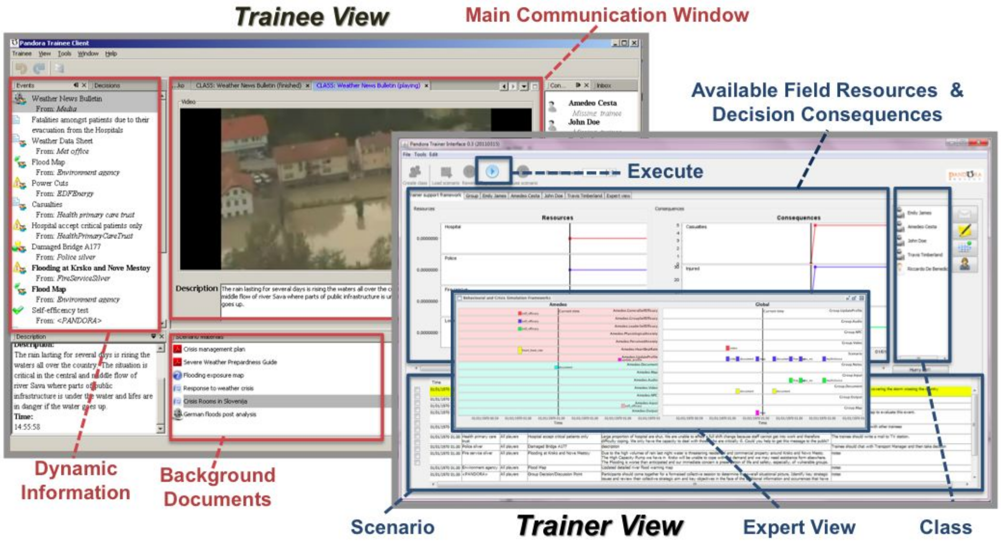
- *trainer-system interaction*, indicated as *Trainer View*, which is related to the functionalities available to the trainer to create a training session, monitor, edit it and interact dynamically with the class;
- *trainee-system interaction*, indicated as *Trainee View*, which is the interface through which the trainee can connect to the PANDORA-BOX, receive stimuli and make decisions about the critical situation.

Additionally we have a further view, called *Expert View*, which is an inspection capability over the timeline environment and its execution functionalities.

Trainer View. This service allows to compose a *training class* completing it with "missing players" to have a coverage of institutional roles in crisis strategic decision making. Created a class the trainer can load a *Scenario*, and see it in tabular form with a series of important information such as the execution time of each goal event and who is the main recipient of information. It is worth highlighting how this representation is close to the current way of working of the trainers and has been instrumental in establishing a dialogue with them, before proposing any kind of completely new solutions. Along with the scenario, the interface also contains information about available resources to resolve the crisis and the consequences of trainees' decisions, both represented through resource timelines and dynamically updated during the training. The trainer is the one to have the basic commands from executing the plan, stopping execution, resuming it and rewinding. Furthermore, a specific requirement from user centered design has been a set of plan annotation functionalities plus a series of additional commands which allows the trainer to dynamically add new stimuli, in perfect line with the mixed initiative interaction style.

Trainee View. The Trainee interface contains three main blocks, plus a number of features related to communication of each trainee with the rest of the class and the trainer. The main building blocks are: (1) *Background Documents*, which represents a set of information delivered off-line to the class in the form of maps, documents, reports, in order to create awareness about the upcoming exercise; (2) *Dynamic information* that represents the information dynamically scheduled and sent to the trainee in the form of videos, maps, decision points etc.; (3) *Main Communication Window*, which is devoted to display stimuli (possibly customized) to individual trainees or to the class.

Expert View. In parallel with the traditional tabular view, the trainer can inspect the more advanced view of the PANDORA module, that is the internal representation for both the Crisis Planner and the Behavioral Reasoner. As already said, all type of information within PANDORA is


Trainee View
Main Communication Window
Available Field Resources & Decision Consequences
Execute
Pandora Trainee Client
Trainee View Tools Window Help
Events
Decisions
CLASS: Weather News Bulletin (finished)
CLASS: Weather News Bulletin (playing)
Video
Description
Scenario materials
Dynamic Information
Background Documents
Scenario
Trainer View
Expert View
Class

Weather News Bulletin
From: *Media*
Fatalities amongst patients due to their evacuation from the Hospitals
Weather Data Sheet
From: *Met office*
Flood Map
From: *Environment agency*
Power Cuts
From: *EDFEnergy*
Casualties
From: *Health primary care trust*
Hospital accept critical patients only
From: *HealthPrimaryCareTrust*
Damaged Bridge A177
From: *Police silver*
Flooding at Krsko and Nove Mestoy
From: *FireServiceSilver*
Flood Map
From: *Environment agency*
Self-efficency test
From: *<PANDORA>*

Description: The rain lasting for several days is rising the waters all over the country. The situation is critical in the central and middle flow of river Sava where parts of public infrastructure is under the water and lifes are in danger if the water goes up.
Time: 14:55:58

Crisis management plan
Severe Weather Prepardness Guide
Flooding exposure map
Response to weather crisis
Crisis Rooms in Slovenija
German floods post analysis

Figure 2: *Screen shots of the current Trainer and Trainee Interfaces*

represented as a timeline and continually updated (see different colors for timelines related to the crisis and the user model in the Expert View). At this point, through the Execute button, the trainer can start the session.

The interaction environment has been critical in our dialogue with the end users and is going to further refined on the one hand to satisfy user requirements on interaction, on the other to make the advanced features more useful for the trainer. Our goal is to fill the gap between the internal representation and users' expectation, with the aim of promoting their active involvement in the management of training.

Conclusions

A first prototype of the complete system has been produced in early December 2010 while a first robust version of the PANDORA-BOX has been officially demoed on March 2011 to the EU project officers during the mid-term project review. This paper shortly introduces this year one demonstrator. It is worth underscoring the important role of planning technology in the PANDORA-BOX.[3] We have seen how the representation with timelines is the core component of the crisis simulation, and that a continuous loop of planning, execution, plan adaptation is created to support personalized training with Trainer in the loop.

Many improvements are scheduled in the remaining lifetime of the project. Just to give an idea, one of the next steps is to provide a tool for Knowledge and Scenario Authoring that allows incremental creation and/or editing of crisis Scenarios. Then, in order to achieve a high degree of realism, stress and pressure, the use of a 3D environment will be explored with the purpose to render a Crisis Room with all trainees together, even if for logistic reasons they are in different locations during the training.

Acknowledgments. The PANDORA project is supported by EU FP7 under the joint call ICT/Security (GA.225387) and is monitored by REA (Research Executive Agency). Authors are indebted to all the project partners for the stimulating work environment.

References

Cesta, A.; Cortellessa, G.; De Benedictis, R.; and Strickland, K. 2011a. Opening the PANDORA-BOX: Planning and Executing Timelines in a Training Environment. In *SPARK-11. Proceedings of the Scheduling and Planning Application Workshop at ICAPS-11, Freiburg, Germany*.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2011b. MRSPOCK: Steps in Developing an End-to-End Space Application. *Computational Intelligence* 27(1):83–102.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th International Conference on Automated Planning and Scheduling*.

Cortellessa, G.; D'Amico, R.; Pagani, M.; Tiberio, L.; De Benedictis, R.; Bernardi, G.; and Cesta, A. 2011. Modeling Users of Crisis Training Environments by Integrating Psychological and Physiological Data. In *IEA/AIE 2011, Part II,*, volume LNAI 6704, 79–88.

Frank, J., and Jonsson, A. 2003. Constraint Based Attribute and Interval Planning. *Journal of Constraints* 8(4):339–364.

Jonsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *AIPS-00. Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling*.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

Pollack, M. E., and Horty, J. F. 1999. There's More to Life than Making Plans: Plan Management in Dynamic, Multi-Agent Environments. *AI Magazine* 20(4):71–84.

Smith, D.; Frank, J.; and Jonsson, A. 2000. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review* 15(1):47–83.

[3] Going back to the title of this short paper: internal to our PANDORA's BOX (**the** `.jar` **in the title**), there is mostly knowledge in terms of timelines and "simple" planning modules for planning and executing them. **Hence planning is our ... *Hope***!!!

Command and Control Training Centers: Computer Generated Forces Meet Classical Planning [*]

Carmel Domshlak
Industrial Engineering & Management,
Technion, Israel

Ziv Even-Zur and Yannai Golany
Elbit Systems, Ltd.
Israel

Erez Karpas
Industrial Engineering & Management,
Technion, Israel

Abstract

We describe SHOGUN, a fully automated system for controlling tactical agents, developed for integration within simulation-based command and control training centers produced by Elbit Systems Ltd. In particular, we focus on describing the action planning module of SHOGUN: while controlling tactical agents in military-style domains involves dealing with uncertainty and partial information in adversarial settings, the planning module of SHOGUN is based on classical, deterministic planning only, and employs a general-purpose classical planner. We describe our embedding of classical planners within the commercial command and control training center, and report on a recent evaluation of SHOGUN in operational scenarios, confronting subject matter experts as trainees.

Introduction

Comprehensive training of forces responsible to react in complex adversarial situations is critical for military high and low intensity conflicts as well as for homeland security scenarios of border control and smart city environments. Such a training should put together teams of trainees at various levels of command, and train them in realistic setups to improve their command and control (C2) capabilities. A vastly dominating portion of C2 training is delegated these days to software simulation systems in which commands of both role-playing trainees and adversary-playing instructors are accomplished by the respective computer generated forces (CGF). Following the paradigm of "train as you fight", the trainees are connected to the virtual battlefield through their operational C2 systems and combat-net radio, coupled by the overall training system to the simulation.

These days, commercial simulations for C2 training already achieve a sufficiently high level of realism in terms of modeling the physical properties of both the environment and forces. The outcome of the training, of course, depends a lot on the effectiveness of the instructors playing the role of the adversary, and this turns out to be an issue. Putting together a team of skilled and coordinated role-players needed for a large-scale simulated exercise requires months of costly preparations, availability of instructors for a long period of time, and a suitable venue. These limitations of relying on human instructors in simulation-based training suggest at least partly replacing them with artificial adversary-players implementing this or another action planning technology. Here we describe SHOGUN, a fully automated system for controlling tactical agents within a commercial military training simulation. SHOGUN has been developed in a joint effort of Elbit Systems Ltd. and the Technion for subsequent integration within the line of large-scale simulation-based training centers produced by Elbit. This system has been recently deployed to Elbit, and successfully passed a detailed performance evaluation.

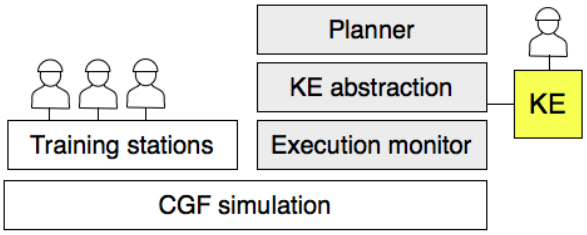
An interesting property of SHOGUN is that the planner it embeds is not just inspired by the artifacts of academic AI research, but actually is such a direct artifact. Moreover, while in general controlling tactical agents in relevant domains involves decision making under uncertainty and partial information in adversarial settings, our experience provides yet more evidence that successful reasoning about real-world systems of active entities does not necessarily have to take explicitly into account all that complexity when choosing between alternative courses of action. While classical planning, capturing single-agent problems with deterministic actions and effectively full knowledge, has been repeatedly criticized for being unrealistic and thus irrelevant to real-world problems, here we demonstrate that this criticism should be taken with lots of caution: The decision making module of SHOGUN is based on classical, PDDL-based planning only, and employs a general-purpose (and thus fully replaceable) classical planner.

In what follows we describe our embedding of classical planners within the commercial C2 training system, as well as the way in which we divide-and-conquer the details of the physical system between the planning and the simulation modules. We then describe the aforementioned evaluation of SHOGUN in operational scenarios, confronting professional military personnel as trainees.

SHOGUN Architecture and Design Decisions

In this section we describe the overall architecture of SHOGUN, focusing on the adopted planning and execution formalism and its support within the system. At high level, SHOGUN comprises a standard architecture of iterative planning, depicted in Figure 1a. It consists of three major modules: (i) a *planning module*, (ii) a plan *execution monitor*,

[*] The work was partly funded by a Magneton Grant. The authors would like to thank Yoav Manor and Gilad Mandel from Elbit Systems for their devoted work on the project.



(a)

```
input: planning task stub $\Pi = \langle V, A, G \rangle$
local: partial-order plan $\rho$
forever:
  receive from EM the current state of knowledge $\sigma$ and
      plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$
  $s$ = TRANSLATE$(\sigma, \Pi)$
  $s'$ = PROGRESSION$(s, \rho_{\text{exe}})$
  if VERIFY-PLAN$(s', \rho_{\text{next}})$ fails then
    $\rho$ = MAKE-PLAN$(\langle V, A, s', G \rangle)$
    send $\rho$ to EM
```

(b)

Figure 1: High-level (a) structure of the system, and (b) flow of the planner interacting with the KE abstraction mapping.

and (iii) a real-time high fidelity 3D tactical computer generated forces (CGF) *simulation* in which the actions selected by both trainees and instructors are actually simulated.

Planning, Execution simulation, and Monitoring

The CGF simulation maintains the entire battlefield arena, and supports an arbitrary number of force types (such as tanks, artillery, reconnaissance, etc.), as long as the simulation is provided with their respective physical models. The simulation runs a full 3D virtual environment of the terrain and physical models of sensors (such as line of sight and detection) and actuators (such as ballistics, path planning, and movement). The battlefield comprises two adversarial forces, blue force and red force, each comprising a, possibly heterogeneous, set of acting units. The trainees fully control the blue force troops and interact with the virtual arena via a training station using a high-level language of command. The planning module replacing the instructors fully controls the red force, and communicates with the simulation via effectively the same language of command. The control of the red force is achieved via a planning and execution loop that takes place during the entire training session. The overall loop is described below and the perspective of the planning module on that loop is pseudo-coded in Figure 1b.

- The execution monitor pulls from the CGF simulation all the data σ required to provide the planner with the current state of the red units (their locations, heading, ammunition, etc.), as well as with those parts of the state of the blue units that are considered by the simulation to be observable by the red units. Status of the blue units not detected by any red unit is not provided to the planner. Likewise, the execution monitor pulls from the CGF simulation a status of the currently executed plan ρ of the red force. Since the execution is continuous, at the moment of the query some of the actions of ρ have been already accomplished, some have started and are still executing, and some are yet to be started. Note that "accomplished" can stand here for both "successfully accomplished" and "failed". In any case, both the collected state of knowledge σ and the plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$ are passed to the planning module.
- The CGF simulation is the core of the virtual arena of Elbit's strategic and tactical training centers, designed to communicate with the training stations of human operators. Hence, the information σ about the current state of the (observable) world takes the form of a raw data. This raw data is then translated to a state of the world description s, corresponding to the abstraction of σ in terms of the planning problem operated by the planner. This translation is based on a knowledge engineering layer that is devoted to bridge between the physical view of the simulation and the symbolic view of the planner.
- Given state s and plan status $\{\rho_{\text{done}}, \rho_{\text{exe}}, \rho_{\text{next}}\}$, the planning module estimates whether the current plan ρ of the red force is still valid. In case the goal of reds turns out to be unachievable from s along the still unaccomplished part of ρ, a new plan is generated from the new initial state s, and passed to the execution monitor.

Classical planner: Why and How.

The heart of SHOGUN is its planning system. The first decision we had to make is whether to develop a special-purpose planner, or to adopt a generic, model-oriented planning system. The second, and in a sense, tangential decision we had to make was what details of the problem the planner should take into account and what details it could ignore without sacrificing the quality of the training.

While in principle special-purpose solutions can be more efficient and effective than generic ones, their development requires the enterprise to establish a development team in the respective area of expertise. Along with the fact that the development basically starts ”from scratch”, that adds numerous risks to the project. Generic planners obviously do not exhibit these risks by the virtue of being generic, having potential to be reused between various verticals. Of course, model-oriented generic planners come with their own risks such as capability of the respective model to capture the desired domain, the computational efficiency of the planner on the domain of interest, etc. However, in contrast to the risks associated with developing a brand new special-purpose system, these risks can be verified in very short time at the beginning of the project using an off-the-shelf planner.

Considering now the choice of the planning formalism, decision making in C2 environments of our interest always involves action non-determinism, partial information, and adversarial settings (Wilkins and Desimone 1992; Tate et al. 2000; Kott et al. 2005). A priori, this suggests that our planning tasks should be specified in terms of much more complicated action models than that of classical planning because the latter assumes deterministic actions, effectively full knowledge, and single-agent setting. Adopting complex planning formalisms, however, comes with a price: the performance of planning for such formalisms currently does not meet the requirements of large-scale C2 training. On the other hand, the performance of classical planners has been dramatically improved over the last two decades, and today these are capable of generating in seconds plans of hundreds of steps in state models of more than 2^{1000} states. In addition, it is of growing understanding that successful reasoning about real-world systems of active entities does not necessarily have to explicitly take into account all the complexity of the reality while choosing between alternative courses of action. This property of many real-world domains has been exploited in the past both in experiments (Yoon, Fern, and Givan 2007; Yoon et al. 2008), as well as in ambitious applications of AI reasoning (Muscettola et al. 1998).

Departing from this matter of business, we have decided to start with a *fully off-the-shelf* satisficing classical planner, adapting it only when really needed and only via external wrappers. Specifically, in the experiments described later on, SHOGUN was using the very popular these days Fast Downward planner (Helmert 2006), using its greedy best first and WA* search engines, and the seminal FF heuristic (Hoffmann and Nebel 2001).

While Fast Downward is based on the SAS$^+$ language (Bäckström and Nebel 1995), which describes a fully deterministic, fully observable, single agent problem, the underlying physical simulation is much more complex. In what comes next we describe our abstraction mapping of planning tasks from the level of simulation to SAS$^+$.

- *Symbolic abstraction of the physical world.* The function TRANSLATE used by the planning module in Figure 1b to map a physical state σ to a SAS$^+$ state s is implemented via a knowledge engineering sub-module (KE). The latter comes to bridge between the general-purpose planner and the specifics of the simulated domain; as such, it is used twofold. First, KE allows a user to define various layers of information over the map. These layers describe strategic points, passable areas, ballistically dominating areas, etc., and for most, they can be derived automatically from the digital map used by the simulation. This processing can be performed once per map, and thus completely offline not only to a specific training session, but to the training in general. In addition, the subject matter expert (SME) in charge of the training session can use KE to further enrich this information by specifying, e.g., regions that he prefers not to be used for movements/positions of specific units. Based on the now defined information layers of the map, KE maps status messages received from the execution monitor to proper values of the respective SAS$^+$ variables. The abstraction of the geographic data such as unit locations and headings is archived via a, possibly non-uniform, grid overlaid on the map.

- *Non-determinism of actions.* While basically all actions of the units are simulated to have stochastic effects, the entropy of the underlying probability distributions is usually low, and typically they have single peaks that take most of the probability mass. A natural abstraction of such actions to fully deterministic SAS$^+$ actions simply ignores all but the most likely outcome of each action. SHOGUN uses precisely that simple abstraction, corresponding to a degenerate form of hindsight optimization, an "online anticipatory strategy" for control problems that has previously been successfully applied to problems of online scheduling (Wu, Chong, and Givan 2002) and probabilistic planning (Yoon, Fern, and Givan 2007; Yoon et al. 2008).

- *Partial observability.* Partial observability in the domain of battlefield training stems from the true modeling of reality in which the information that is available to the planner is only what the red force "sees": blue units which are not detected by any red units are not reported to the planner. We use "optimistic sensing" to get rid of this partial observability as follows: when a red unit performs a sensing action (that is, looks in some direction, trying to find blue units) the expected effects of that action are that no blue forces will be detected. If there are indeed no blue forces - the plan can proceed normally. If there are blue forces there, then the current plan is most likely no longer valid, and therefore re-planning is performed, this time accounting for the "new" blue forces.

- *Optimization objectives.* In most battlefield scenarios, the mission is to achieve some objective, while trying to minimize friendly losses. Since we use single-agent planning, we do not directly account for enemy actions, and specifically, we do not plan for friendly units to be destroyed. Therefore, we do not directly try to minimize friendly losses, but rather try to minimize *risk*. We associate a risk level with each action, by assigning higher costs to riskier actions. For example, maneuvering in a flat area at the base of an enemy-occupied hill is riskier than maneuvering on top of a hill, and is therefore more expensive. Although the planner we use is not an optimal planner, it does try to find a low-cost plan, which directly translates to a low-risk plan.

Domain Formulation

Several domain formulation choices affect the entire system. First, as stated before, we divide the map into locations, which are arranged on a grid, where each location can hold multiple friendly units, and multiple enemy units. Each grid location is represented by an object in the planning problem, and thus locations are used as parameters for operators and predicates. The translation of world knowledge to a planning state involves mapping units at specific coordinates to the corresponding grid locations.

Second, entities in operational domains often act in line with some standard operating procedures (SOP), and thus, in particular, act in *formations*. In maneuvering, for instance, a formation could be either a single entity moving by itself, 2 entities moving side-by-side, 3 entities moving in a single column, or any other arrangement. Types of formations are defined by the overall set of SOPs, and can be provided by a subject matter expert. We chose to formulate our domain so that all actions are performed by some formations of entities. For example, moving from one location on the grid to a neighboring location is done by using the *Move* action on a formation, which describes the entities to be moved, and their internal arrangement (side-by-side, column, etc.). Two special types of action, *Set-Formation* and *Break-Formation*, allow entities to rearrange themselves in different (possibly larger or smaller) formations. Note that this is similar in spirit to the well-known Logistics planning benchmark from IPC-1998 and IPC-2000, where a formation can be thought of as a truck, and an entity can be thought of as a package loaded into the truck (aka joining formation). This engineering methodology appears to be quite useful in general; for instance a very similar technique has been used by Balla and Fern (2009) in their recent work on action selection for tactical assault, evaluated by the authors on Wargus computer games.

Third, we had to deal with enemy units on the battlefield, and their partial observability. We model enemy presence by creating a state variable for the number of enemy units at each grid location. The possible values for this range are either a number (between 0 and some bound) or *unknown* - a special value indicating that we have no knowledge about enemy presence in that grid location. Thus, the (expected) effect of performing a sensing action on a given grid location is that if the number of enemy units in that location was *unknown*, it becomes 0, and otherwise, there is no effect. This formulation also allows us to ignore the identities of enemy units, which are not part of the knowledge provided to the planner anyway.

Load Balancing and Parallelization

One addition to the standard classical planning setting that we found essential was load balancing between the red units. At high level, the load balancer in SHOGUN pre-assigns each sub-goal to a subset of units, decomposes the overall planning task into several smaller tasks that are planned for independently, and then combines their solutions into a plan for the overall task. This procedure is important for balancing the workload between different role-players, causing forces to act in a more "coordinated matter", and reduces the size of the individual planning tasks solved by the planner.

Similarly to all other system components, the load balancer in SHOGUN is completely domain independent. It starts by assigning to each goal a subset of units that can achieve it as cheaply as possible in the (easy to solve) delete-relaxed version of the planning task. Then, the rest of the units are assigned in proportion to the cost of the relaxed plan for each of the goals, so that goals that are riskier to achieve are assigned more units. Finally, the goals are grouped based on the transitive closure of the forces assigned to them, and each such group of goals is planned for using only the forces assigned to it. In the domain considered here, plan combination is trivial, since no two plans can interfere with each other.

One thing to note is that, although the load balancer might assign several units to a single goal, the planner might still not utilize all of these units (the plan found could involve just a single unit). In order to force SHOGUN to act more realistically, we artificially increase the cost of action repetitions. This puts a heavy bias toward using more than a single unit in each plan, resulting in plans allocating forces to goals in ad hoc proportion to the size of the goal.

Finally, though the quality metric for our plans is risk reduction, and thus we employ a cost-oriented, sequential planning, the actions of different units often can (and if so, should) be applied concurrently. While we do not plan for this second objective directly, we do convert our sequential plans into partial order plans allowing concurrent action execution. If our domain had been formulated in plain STRIPS, then simple Partial Order Causal Link (POCL) backward analysis of the plan would have given us a desired partial order. However, our domain formulation uses conditional effects, and this requires slight extension of the standard POCL analysis. To establish hypothetical causal relations between the actions, we first simulate sequential execution of the initial plan, determine which conditional effects of which action instances along the plan have been fired, compile the fired conditional effects into now unconditional effects of the respective actions, and then perform the standard POCL backward analysis of the plan. The resulting parallelization is sound and complete, and results in realistic schedule of plans for our multi-unit forces. Overall, the simultaneous acting effect, achieved in SHOGUN via the mixture of load balancing and plan parallelization, achieves the effect of a standard military C2 methodology called "mission-oriented C2", allowing for solving large-scale problems by wisely delegating its sub-parts to different planners.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Comp. Intell.* 11(4):625–655.

Balla, R., and Fern, A. 2009. UCT for tactical assault planning in real-time strategy games. In *IJCAI*.

Helmert, M. 2006. The Fast Downward planning system. *JAIR* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Kott, A.; Budd, R.; Ground, L.; Rebbapragada, L.; and Langston, J. 2005. Building a tool for battle planning: Challenges, tradeoffs, and experimental findings. *Applied Intelligence* 23(3):165–189.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote Agent: To boldly go where no AI system has gone before. *AIJ* 103(1-2):5–47.

Tate, A.; Levine, J.; Jarvis, P.; and Dalton, J. 2000. Using AI planning technology for army small unit operations. In *AIPS*.

Wilkins, D., and Desimone, R. V. 1992. Applying an AI planner to military operations planning. In *Intelligent Scheduling*, 685–709. Morgan Kaufmann.

Wu, G.; Chong, E. K. P.; and Givan, R. 2002. Burst-level congestion control using hindsight optimization. *IEEE Transactions on Automatic Control* 47(6):979–991.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A baseline for probabilistic planning. In *ICAPS*, 352–359.

Constraint Priorities - a Way of Getting an Optimal Timetable Fully Automatically. Three Steps Into the Modern Timetable Scheduler

Wieslaw Dudek

Wieslaw Dudek Timetables, Krakow, Poland
Wieslaw.Dudek@gmail.com

Abstract

The presentation goes through all the system features which make the new timetabling technology unique and efficient, presenting constraint priorities as the most important innovations in the scheduling process. The system of priorities exists in the main three fundamental modules: inconsistency detection system, optimization module and solver. The complete constraint system is also presented as an important part of useful timetabling software.
Moreover, the presentation covers other features which can also make the solution useful for other non-school problems. The described solution is available at www.school-timetable.eu.

Introduction

The process of building a timetable is very complex and arduous as it requires many difficult constraints to be reconciled at once. This is too great a challenge for people even in cases of small bundles of requirements but can be simplified by using modern software systems.

In this paper the modern system means software which can take care of the whole scheduling process without bothering its users too much and without asking any questions about how to arrange some classes which apparently had been too difficult for the system to set. All the questions should be constructive ones.

The www.school-timetable.eu site is such a new, modern solution to the problem. The most innovative feature is prioritizing the constraints. It is so natural for people working on a timetable to abandon some unimportant requirements due to lack of time or skills. Time and skills count in virtual reality as well.

Constraint priorities are taken into account in the main three fundamental modules in the system i.e. inconsistency detection system, optimization module and solver.

Every scheduler needs a complete set of requirements, which can be defined by the user, to achieve the goal of getting a timetable in a fully automatic way, simply by clicking on „Generate” button. Without a complete set, the received solution would be imperfect and would require user interaction and analysis; in some cases the whole timetable would need to be rebuilt making the solution useless.

If users are given a possibility to enter all of their requirements into the timetable, they most likely enter all of them and thus schedulers need good inconsistency and optimization systems based on priorities.

Inconsistency Detecting System

The inconsistency detecting system filters the requirements and removes the ones with less priority which are in conflict with the important ones. The priorities can be assigned to constraints based on their category i.e. min, max quantity, gap mode etc. and the resource they concern e.g. a teacher, a classroom.

Assuming the priorities were assigned to constraints correctly, the system will erase only the ones that must be removed i.e. the ones with lower priorities.

Solver

As soon as the inconsistency detecting system removes inconsistencies, the solving process begins. This process needs to be efficient and it is crucial for schedulers to consider it effective. The www.school-timetable.eu system has got the most powerful solver. The solver was tested on data coming from 2008-2010. The 4.0 version proves its effectiveness by arranging *all* of the classes with good execution time, provided the user turned on the correction process or has not banned any of the constraints from being removed during the optimization process. The algorithm managed to solve a timetable with 13 sites, 1300 courses, 12000 subjects, 30000 students in it, which means it can be used for timetabling a university schedule with shared classrooms, teachers etc.

The solver takes into consideration each requirement which needs to be defined for school timetables and can be easily extended with new ones. At present the solver copes with all of the constraints used in school planning that were necessary during the last few years of the system development, even at times when full timetables needed to be built without enough information provided e.g. missing classes, unknown language levels.

The quest for completeness produced flexibility of the system and now it is able to handle other types of timetables e.g. work schedules. It is worth mentioning such constraints, which are not present or very rare in other timetabling systems, like different working times for sites, inter-moving modes and times, classes order, first/last lesson in a day, packages of courses for individual students, resources other than classrooms, combined & correlated classes, shift work, space between classes etc.

Optimization Module

Often to get a complete solution some systems need their users to manually abandon some difficult constraints and decide which ones need to be removed first. Our system is much more „aware" of timetabling possibilities and reality than its users; everything it needs from them is the importance of constraints in a form of their priority. There is no reason to ask the users to help solve some sub-problems because the computer has much more chance to try them out quickly.

The optimization process is a way of getting a complete solution. The last step in it is the correction process, which is the last resort to receive a full solution if the user had not allowed to remove some of the constraints (called fixed) because of their high importance. The correction process works in two runs. The first run will be called here „the ambitious run" and results in a fully or partially built timetable with all important (fixed) constraints in their place. The second run, called „the good enough run," arranges the rest of the classes which were not able to be arranged in the previous „ambitious run." However, this time without constraint fixing so it will be possible to remove them if necessary and place all the remaining classes. This is modeled after people who need to abandon their ambitious plans before they fail; doing so at the very last moment they are able to get much further, had they taken a shortcut right at the beginning.

The timetable correction is fully automatic but can also be used manually if the users want to. It also lets its users fix more constraints than it is really needed if they are afraid they could be removed by the system too early.

System Architecture

The ideas mentioned in the preceding paragraphs were foundations of the www.school-timetable.eu system architecture depicted on Figure 1.

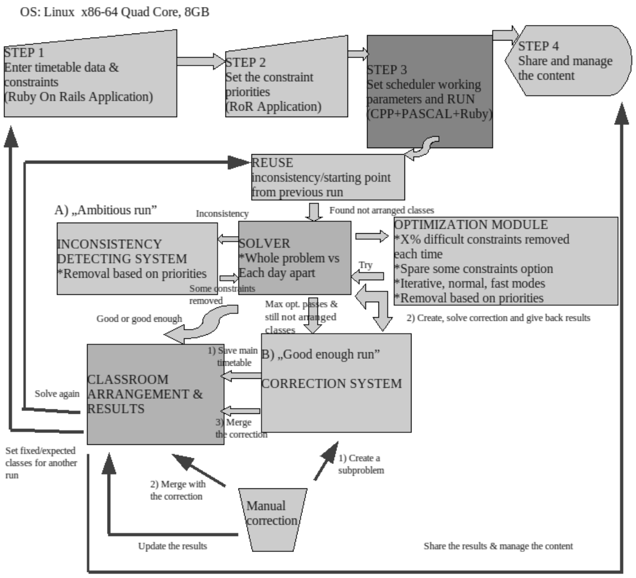

OS: Linux x86-64 Quad Core, 8GB
STEP 1
Enter timetable data & constraints
(Ruby On Rails Application)
STEP 2
Set the constraint priorities
(RoR Application)
STEP 3
Set scheduler working parameters and RUN
(CPP+PASCAL+Ruby)
STEP 4
Share and manage the content
REUSE
inconsistency/starting point from previous run
A) „Ambitious run"
Inconsistency
Found not arranged classes
INCONSISTENCY DETECTING SYSTEM
*Removal based on priorities
SOLVER
*Whole problem vs Each day apart
OPTIMIZATION MODULE
*X% difficult constraints removed each time
*Spare some constraints option
*Iterative, normal, fast modes
*Removal based on priorities
Try
Some constraints removed
Max opt. passes & still not arranged classes
Good or good enough
2) Create, solve correction and give back results
CLASSROOM ARRANGEMENT & RESULTS
1) Save main timetable
B) „Good enough run"
CORRECTION SYSTEM
3) Merge the correction
Solve again
Set fixed/expected classes for another run
2) Merge with the correction
1) Create a subproblem
Manual correction
Update the results
Share the results & manage the content

Figure 1: Overview of the www.school-timetable.eu system architecture

As one can see from the process flow perspective showed on Figure 1, the system fully automatically applies a wide range of techniques in order to receive a complete solution. There is no need for difficult questions like "How to arrange some classes?" because the system can decide by itself which difficult constraint it should remove on the basis of constraint priorities. However, a user is allowed to change the priorities dynamically by creating corrections (see „Manual correction>Create a subproblem") or even arbitrarily impose a given distribution of classes manually (see „Manual correction>Update the results").

Besides, it is possible to define an expected or fixed distribution of classes at the very beginning if one is aware of all of the requirements at this stage. Taking a closer look into the remaining steps of the process we can distinguish the following ones: entering data /constraints defining/, assigning priorities to constraints and sharing and managing the final timetable.

Constraints Defining

The step of entering data needs to be sufficient and effective. Sufficiency means a complete set of constraints which can be reflected in the system (see „Solver" chapter) while effectiveness signifies a simplicity of applying constraints. For many scheduling systems defining groups of students seems to be the most difficult thing to do.

Creating groups is related to many reasons; one of them concerns student body sizes - some of them are too big to be taught, other ones suffer from the shortage of teachers. Another reason for dividing students into groups can be a need to combine groups on student body, year or school level. At the end students can select an individual education mode or some optional courses. The system has to be on alert and ensure that some common constraints for such groups will be met e.g. to avoid overlaping of courses selected by the same student, impose no gaps for students in certain kinds of schools or make sure that some defined constraints for the groups will be met.

In www.school-timetable.eu scheduler system the way of defining groups depends on the initial data we have. If we already know the way of dividing students into groups, the whole process can be simplified by using packages of courses (see Figure 2).

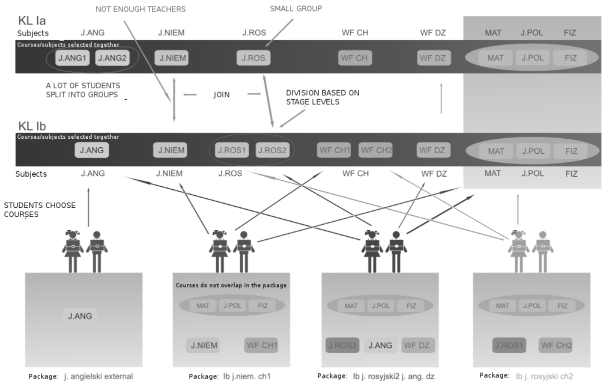


Figure 2: Defining the packages of courses

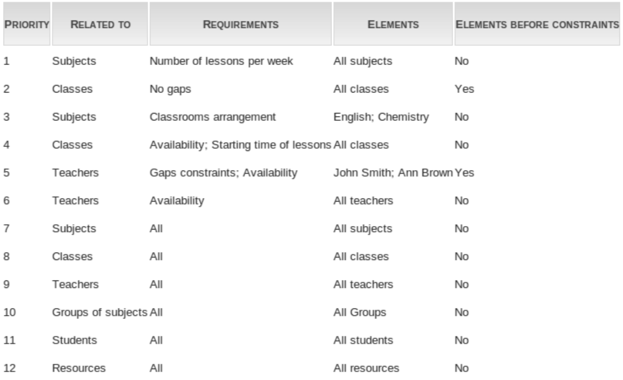
Sometimes it is impossible to say how students are going to be divided, nonetheless, we need to secure a certain selection of groups avoiding overlaping of courses; later on appropriate courses will be selected according to skills or preferences. To achieve this goal we cannot use packages of courses but need a more flexible solution - correlating subjects designed by a maximum number of simultaneous lessons from a group of subjects or a subject combining feature.

Setting a minimum number of simultaneous lessons can be a way of correlating some classes of subjects; however, setting a max number of simultaneous lessons to one could be a way of separating subjects in time. On the other hand, the goal of subject combining is to correlate lessons, share a classroom by some of the subjects, define a common lesson for several classes, define elective line i.e. a block of many classes of many subjects where each student may choose one subject from that line.

Priorities Defining

The second step of the flow in Figure 1 is to assign priorities to the previously entered constraints. To simplify the assignment process there are some predefined priorities for existing constraints which do not have to be redefined in many cases. However, if necessary, they can be changed e.g. a user can demand no gaps for a few teachers as a priority request.

In the example below (Figure 3) John Smith and Ann Brown's teacher gaps & availabilities are set to be more important than the same constraint types for other teachers. Also John Smith's gaps & availabilities are more important that the same ones for Ann Brown because "Elements before constraints" attribute is set to "Yes". With the "No" value the order would be different: John Smith's gaps > Ann Brown's gap > John Smith's availability > Ann Brown's availability.

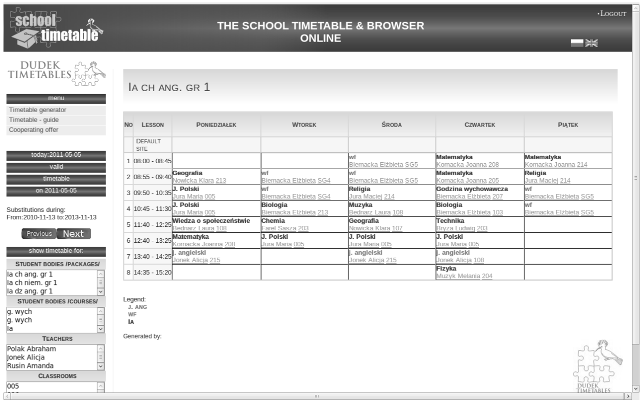


PRIORITY	RELATED TO	REQUIREMENTS	ELEMENTS	ELEMENTS BEFORE CONSTRAINTS
1	Subjects	Number of lessons per week	All subjects	No
2	Classes	No gaps	All classes	Yes
3	Subjects	Classrooms arrangement	English; Chemistry	No
4	Classes	Availability; Starting time of lessons	All classes	No
5	Teachers	Gaps constraints; Availability	John Smith; Ann Brown	Yes
6	Teachers	Availability	All teachers	No
7	Subjects	All	All subjects	No
8	Classes	All	All classes	No
9	Teachers	All	All teachers	No
10	Groups of subjects	All	All Groups	No
11	Students	All	All students	No
12	Resources	All	All resources	No

Figure 3: Example of constraint priorities defining

Sharing and Managing the Final Timetable

Ultimately, after a timetable is created, it can be easily accessed online by students, classes, teachers & classrooms supervisors. If required, registering for courses can be turned on and even more preferences can be taken into account during the scheduling process. Another option for schools is the possibility to manage teacher substitutions online. The change will be immediately communicated to the end users /students, teachers etc./.



Substitutions during:
From:2010-11-13 to:2013-11-13

Student bodies /packages/: Ia ch ang. gr 1, Ia ch niem. gr 1, Ia dz ang. gr 1

Student bodies /courses/: g. wych, g. wych, Ia

Teachers: Polak Abraham, Jonek Alicja, Rusin Amanda

Classrooms: 005

Ia ch ang. gr 1

No	Lesson	Poniedziałek	Wtorek	Środa	Czwartek	Piątek
	Default site					
1	08:00 - 08:45			wf Biernacka Elżbieta SG5	Matematyka Kornacka Joanna 208	Matematyka Kornacka Joanna 214
2	08:55 - 09:40	Geografia Nowicka Klara 213	wf Biernacka Elżbieta SG4	wf Biernacka Elżbieta SG5	Matematyka Kornacka Joanna 205	Religia Jura Maciej 214
3	09:50 - 10:35	J. Polski Jura Maria 005	wf Biernacka Elżbieta SG4	Religia Jura Maciej 214	Godzina wychowawcza Biernacka Elżbieta 207	wf Biernacka Elżbieta SG5
4	10:45 - 11:30	J. Polski Jura Maria 005	Biologia Biernacka Elżbieta 213	Muzyka Bednarz Laura 108	Biologia Biernacka Elżbieta 103	wf Biernacka Elżbieta SG5
5	11:40 - 12:25	Wiedza o społeczeństwie Bednarz Laura 108	Chemia Farel Sasza 203	Geografia Nowicka Klara 107	Technika Bryza Ludwig 203	
6	12:40 - 13:25	Matematyka Kornacka Joanna 208	J. Polski Jura Maria 005	J. Polski Jura Maria 005	J. Polski Jura Maria 005	
7	13:40 - 14:25	j. angielski Jonek Alicja 215		j. angielski Jonek Alicja 215	j. angielski Jonek Alicja 108	
8	14:35 - 15:20				Fizyka Muzyk Melania 204	

Legend:
J. ANG
WF
IA

Generated by:

Figure 4: Sharing the timetable online

Development Perspectives

The author's desire is to build a modern and user-friendly tool for scheduling purposes both for schools and other institutions working with timetables; the system should be capable of creating optimal timetables and should allow its users to access them online. The release of the Android version of the timetable browser is planned in 2011. The new interfaces for institutions other than schools should be soon put into place.

Although the algorithm is in its final state of development there are still opportunities for improvement. Generating a timetable for a university with all of its departments is available, though the whole process takes a few hours. It is possible to increase its speed by using concurrent threads and splitting techniques.

Another area for development is an classrooms arrangement system which could be capable of operating on groups of classrooms instead of individual ones. It is worth mentioning the another area for improvement such as integration with other systems. Although the system is ready to operate on the open JSON RPC standard and it is also fast enough to be used separately, the integration with other existing systems can be convenient for the end users. It is possible to develop some pieces of administrative software to use with the timetable generator or to integrate it closely with other systems which perform their tasks well.

History

The idea of the scheduler in its current state has been maturing for 16 years since 1995 when it was first taken into consideration. The year 2008 turned out to be critical for the project since the main questions were answered and a new online system was built.

However since its first release in 2008 many parts of the generator have been improved. Here are some of the most important dates in the project's development. Most of the following tasks were executed by one person only - the author:

- 1995-2008 - idea development,
- IV 2008 - the beginning of the project and its first release - v1.0,
- V-VI 2008 - improvements in inconsistency system,
- VI 2008 - gap constraints revised,
- VII 2008 - classroom management system (preferred classrooms, arrangement modes, fixed classrooms) & no 1 or 2-hour-long working days,
- IX 2008 - student groups, combined subjects, more improvements in inconsistency system,
- II 2009 - v2.0 with new graphical design and lots of improvements: sharing & managing timetables online (substitutions); priorities used in inconsistency system; new important constraints introduced: group of days, order of classes, sites with independent set of times of lessons, blocked subjects, students & courses,
- VII 2009 - optimisation system using priorities,
- XI 2009 - v3.0 with usability improvements, packages of courses, registration for courses before or after scheduling, correlation or separation of subjects within subject groups,
- XII 2009 - optimisation system improved,
- III 2010 - v4.0 performance & efficiency improved; JSON-RPC & XSD,
- VIII 2010 - successful execution of large university timetables with up to 13 sites and 1300 courses with about 10 subjects each (12000 subjects were defined),
- XII 2010 - new usage scenarios and application - work scheduling; data entering speed-up and simplicity - mass change & automatic package of courses generation based on courses properties,
- I 2011 - bilingual version and foundation for multilingual version; exports & imports of data,
- II 2011 - timetable correction by days and further speed-up of generation process,
- III 2011 - timetable correction by sub-problem /solving sub-problem in independent way as an external timetable and merge results afterwards/, automatic correction if not arranged lessons; manual and arbitrary corrections.

References

Demo samples: http://www.school-timetable.eu/access/demo

A service guide: http://www.school-timetable.eu/guide/guide

A service tutorial: http://www.school-timetable.eu/access/tutorial

More info: http://www.school-timetable.eu

Planning Multi-modal Transportation Problems

José E. Flórez
jflorez@inf.uc3m.es

Álvaro Torralba Arias de Reyna
alvaro.torralba@uc3m.es

Javier García
fjgpolo@inf.uc3m.es

Carlos Linares López
carlos.linares@uc3m.es

Ángel García-Olaya
agolaya@inf.uc3m.es

Daniel Borrajo
dborrajo@ia.uc3m.es

Computer Science Department, Universidad Carlos III de Madrid
Avenida de la Universidad 30, 28911 Leganés, Madrid, Spain

Abstract

Multi-modal transportation is a logistics problem in which a set of goods have to be transported to different places, with the combination of at least two modes of transport, without a change of container for the goods. The goal of this paper is to describe TIMIPLAN, a system that solves multi-modal transportation problems in the context of a project for a big company. In this paper, we combine Linear Programming (LP) with automated planning techniques in order to obtain good quality solutions. The direct use of classical LP techniques is difficult in this domain, because of the non-linearity of the optimization function and constraints; and planning algorithms cannot deal with the entire problem due to the large number of resources involved. We propose a new hybrid algorithm, combining LP and planning to tackle the multi-modal transportation problem, exploiting the benefits of both kinds of techniques. The system also integrates an execution component that monitors the execution, keeping track of failures and replans if necessary, maintaining most of the plan in execution. We also present some experimental results that show the performance of the system.

(The full paper describing this system appears in the ICAPS-11 Proceedings at pages 66-73)

PLANET : a planning and replanning tool for a constellation of agile Earth-observing satellites

Romain Grasset-Bourdel[⋆†] and **Gérard Verfaillie**[⋆]
[⋆] Onera - The French Aerospace Lab, Toulouse, France
{Romain.Grasset, Gerard.Verfaillie}@onera.fr

Antoine Flipo[†]
[†] CNES, Toulouse, France
Antoine.Flipo@cnes.fr

Abstract

We present the problem of planning off-line on the ground all the activities of a constellation of next-generation agile Earth-observing satellites and the specific algorithm that was developed to solve it. Then, we present the replanning problem that arises when urgent observation requests are received during plan execution. We show how the planning algorithm can be used in this replanning setting, with some modifications that limit computing time and favour plan stability and optimality. We finally introduce PLANET as a tool based on these algorithms, and demonstrate algorithm efficiency.

Motivation

The context of the work we present in this paper is the European defence MUSIS project (Multinational Space-based Imaging System for Surveillance, Reconnaissance, and Observation) and more precisely the management of the MUSIS agile satellites that are equipped with high-resolution optical observation instruments.

As usual, such satellites are managed from the ground by a mission planning system which receives user observation requests, builds regularly satellite activity plans over a limited horizon ahead (typically one day), and receives plan execution reports. These plans must meet all the physical constraints and satisfy as well as possible the user requests.

However, such a management system is not very reactive. Any observation request, arriving at any time during the day, must wait for the next day to be taken into account. This led project managers to consider a more reactive management system that would take full advantage of the presence of several ground control stations and of the numerous associated satellite visibility windows that allow updated activity plans to be uploaded.

In such a setting, replanning may be called before any satellite visibility window. Replanning problem data is, on the one hand, a current activity plan involving hundreds of observations and, on the other hand, some urgent observation requests (at most some tens). The goal is to build quickly (efficiency) a new plan over the rest of the day that is of an as high as possible quality (optimality) and is as close as possible to the previous one (stability).

Planning problem

The constellation we consider is made up of two identical satellites[1] moving on the same orbit (circular, low altitude, quasi-polar, and heliosynchronous) with a phase shift of 180 degrees between the two satellites.

Each satellite is equipped with thrusters for potential orbital manoeuvres and gyroscopic actuators for quick attitude movements, useful to perform observations and transitions (Lemaître et al. 2002).

A telescope, with two focal planes, allows observations to be performed in the visible and infra-red spectra, with two images (visible and infra-red) within day periods (on the ground) and only one image (infra-red) within night periods. A mass memory allows observation data to be recorded and a high-rate large-aperture antenna allows it to be downloaded towards ground reception stations. Solar panels allow batteries to be recharged when the satellite is not in eclipse. For the sake of agility, all these equipments are body-mounted on the satellite.

We do not go into details here, but the numerous physical constraints that must be met can be classified into six classes : attitude trajectory, observation, download, memory, instruments, and energy. Some of them are similar to the thermal and pointing constraints considered in (Chien et al. 2010) for scheduling operations on board EO-1.

With each user request, are associated a polygon which has been split into strips, a priority level, a weight, and a deadline. Typically, three priority levels are available, from 3 to 1. It is assumed that any request of priority p is preferred to any set of requests of priority strictly less than p. Weights allow to express preferences between requests of the same priority level and are assumed to be additive.

It is assumed that any strip can be observed using only one strip overflight. With each strip, are associated a geographical definition, observation durations (day or night), image sizes (visible, day or night infra-red), a maximum observation angle, and a set of triples ⟨satellite, visibility window, weather forecast⟩.

[1] The planning algorithm we propose is able to manage any number of satellites, possibly not identical: not the same parameter values.

User requests may arrive at any time and, each day, at a given time, a plan is built for the next day from all the requests that are not out of date and not fully satisfied yet. This plan is built on the ground and then uploaded to the satellites for execution. Typically, up to ten minutes of computing are available for planning. After plan execution, observation data that has been downloaded to the ground is analyzed, taking into account the actual cloud cover, and satisfied requests are removed.

In addition to these normal user requests, urgent ones may arrive at any time too. The latter must be taken into account as soon as possible. To do that, before any visibility window between a ground control station and a constellation satellite, an updated plan is built for the rest of the day from all the requests, either normal or urgent. Replanning is guided by two objectives: on the one hand, to produce a new plan of highest quality, as in planning, and, on the other hand, to maintain in the new plan the greatest number of observations present in the previous one, because a plan is a kind of commitment facing users. In order to be able to take into account urgent requests until the last minutes, we consider that half of the computing time available for planning is available for replanning, that is up to five minutes.

The planning problem can be modeled using for each satellite the following state variables: the current time (orbital position); the attitude position and speed along the three axes; the available memory and energy; for each instrument, its status (ON or OFF), the remaining ON time, and the remaining number of ON/OFF cycles; for the antenna and the visible focal plane, its temperature.

Six types of action are available for each satellite: orbital manoeuvres which are mandatory, observations, data downloads, heliocentric pointings, geocentric pointings, and instrument switchings.

It must be observed that actions of all the types, but the third and sixth (data downloads and instrument switchings), constrain the satellite attitude and are thus mutually exclusive. They must be performed in sequence. Only data downloads and instrument switchings can be performed in parallel, at any time for instrument switchings, but only within effective communication windows for data downloads. As a consequence, a plan has the form of a sequence of actions of any type, except the third and sixth, with attitude movements between consecutive actions and with data downloads and instrument switchings in parallel.

The criterion to be optimized is a vector of numbers v_p, one for each priority level p. Two vectors resulting from two plans are lexicographically compared. For each priority level p, v_p is the sum of the weights of the requests r of priority p, weighted by four factors whose value is between 0 and 1 and which represent (1) the percentage of realization (observation and data download), (2) the mean percentage of cloud cover, (3) the mean observation angle, and (4) the mean data delivering delay, over all the strips of the polygon associated with r.

Planning algorithm

To solve this planning problem, we developed a specific chronological forward search algorithm with dedicated decision heuristics, constraint checking, limited lookahead, and backtrack in case of constraint violation, which guarantees the production of a plan that may be not optimal, but is really executable by the satellites.

Decreasing priorities First, the algorithm we developed works by decreasing priority levels from 3 (the highest) to 1 (the lowest). We consider the sequence of observations present in the plan produced at level $p+1$ as being mandatory (without fixed starting times) when building a plan at level p. Such an approach is justified by the fact that any request of priority strictly greater than p is preferred to any set of requests of priority p.

A forward chronological algorithm At each priority level p, the algorithm builds a plan in a forward chronological way, from the beginning Ts of the planning horizon to the end Te. At each algorithm step (see Figure 1), if t is the current time and o is the next mandatory observation to be performed, the algorithm chooses the next observation o' of priority p to be performed before o (o if no such observation exists). The algorithm stops when there is no other observation to be included in the plan.

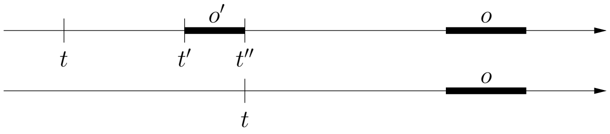


Figure 1: At each algorithm step, choice of the next observation to be performed: o is the next mandatory observation, o' is the chosen observation of priority p.

Decision levels This choice of the next observation to be performed is the first algorithm decision level. Once it has been made, the algorithm makes other choices over the temporal horizon from t to t'' (see Figure 2) at other decision levels: (2) possible insertion of geo or heliocentric pointings, (3) possible data downloads, and (4) instrument activations.

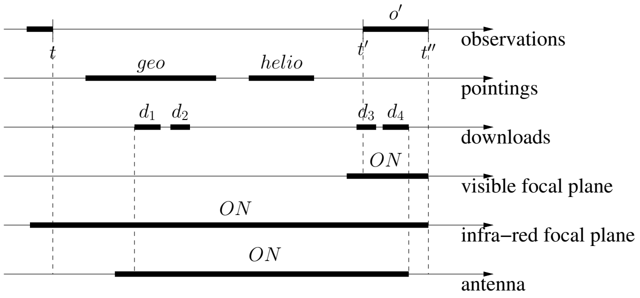


Figure 2: Example of decisions at the four levels: (1) observations, (2) pointings, (3) downloads, and (4) instruments.

Once decisions have been made at the four levels, a consistent plan is available from t to t'', extending the plan that already exists from Ts to t, and the planning process can continue from t'', starting from a known satellite state.

This incremental process, which builds incrementally a complex system trajectory, is the main justification for using a forward chronological search.

For the sake of simplicity, we present the algorithm by assuming only one satellite. However the planning process is in fact interleaved on the two satellites and the next planning step is the earliest one over the two satellites.

Backtracks At any decision level, in case of constraint violation, other choices are made. If no other choice is available, a hierarchical backtrack at the relevant decision level is triggered (see Figure 3).

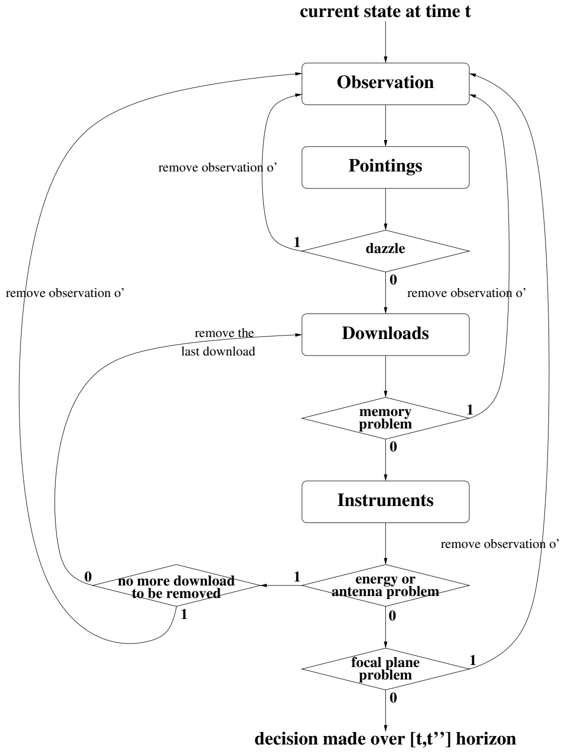


Figure 3: Hierarchical backtracks between decision levels.

At the first level, if the chosen observation is a mandatory one (higher priority), and the insertion is impossible, a chronological backtrack is triggered to the previous insertion of an observation of priority p. However, in order to avoid as much as possible such situations, the latest observation ending times of mandatory observations are propagated from the end to the beginning of the sequence before planning.

Heuristics Heuristics are necessary to make choices at all the decision levels. These heuristics are crucial to the production of good quality plans because, for the sake of efficiency, the algorithm backtracks only in case of constraint violation and never to try and improve on the current plan.

The implemented heuristics are not detailed here.

Dealing with replanning

The first question is how to define stability and how to combine quality and stability (Fox et al. 2006).

In our problem, the quality of a plan is measured by a vector of utilities v_p, one for each priority level p. We maintain this global hierarchical view when replanning. For each priority level p, let R_p be the set of requests of priority p. For each request r, let w_r be the utility associated with r. We have: $v_p = \sum_{r \in R_p} w_r$. Let $I_p \subseteq R_p$ be the set of requests r of priority p that are negatively impacted by replanning (at least one strip of the polygon associated with r was present in the previous plan, but does not appear in the new one). We define the stability as the sum over the impacted requests of the loss in utility: $s_p = \sum_{r \in I_p} (w'_r - w_r)$ with w'_r (resp. w_r) the previous (resp. new) utility associated with r. s_p is positive or null. The lower s_p, the more stable the plan. Then, we define the criterion to be optimized when replanning as a weighted combination of quality and stability: $vs_p = v_p - \alpha.s_p$, with α a positive parameter to be set by system users according to the importance they attach to stability with regard to intrinsic quality.

The data of a replanning problem is very similar to the one of a planning one: same requests, state variables, actions, and constraints. The main difference is in the definition of the criterion to be optimized. Specific data is however: the previous plan, a set of urgent requests to be taken into account and, for each constellation satellite s, a replanning horizon.

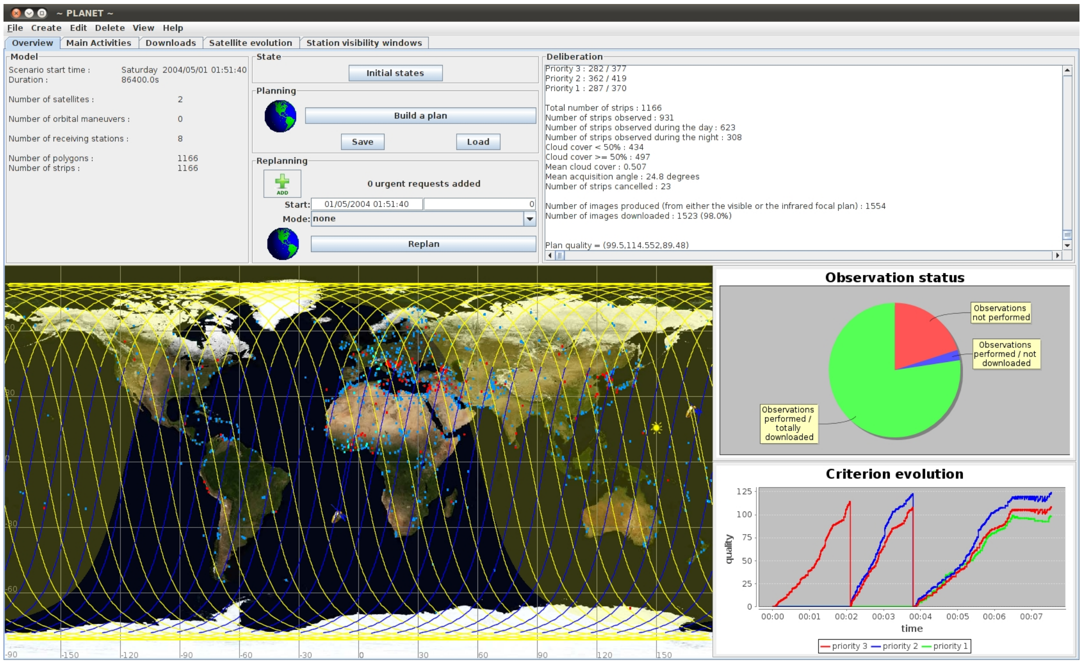
To solve our problem, we did not choose to use local search methods, mainly because of the high potential cost of a local change: adding or removing an action in the middle of a plan requires the complex system trajectory to be computed and checked again from the adding/removing point to the end of the planning horizon. We chose to use for replanning the same forward chronological search algorithm we used for planning, called with slightly different data.

We consider four possible modes of replanning. Roughly speaking, the search is less and less restrictive from the first to the fourth mode: less and less constraints imposing previously planned observations (by modifying priorities or weights in heuristics), more and more observations taken into account. It would be possible to run these modes sequentially or concurrently and to get the best result obtained by the deadline.

PLANET

Planning and replanning algorithms were implemented in a tool, called PLANET for PLanner for Agile observatioN satElliTes (see Figure 4), which was developed for this mission, on the basis of a previous tool (Beaumet, Verfaillie, and Charmeau 2011).

Algorithms were experimented on a real-size realistic instance, built by CNES (French Space Agency) and whose characteristics are the following ones: a one-day planning horizon; 8 ground reception stations; 3 priority levels; 1166 observation requests; all of them with polygons limited to one strip and all of them of the same weight (1); among



Model

Scenario start time : Saturday 2004/05/01 01:51:40
Duration : 86400.0s

Number of satellites : 2

Number of orbital maneuvers : 0

Number of receiving stations : 8

Number of polygons : 1166
Number of strips : 1166

0 urgent requests added

Start: 01/05/2004 01:51:40 0
Mode: none

Deliberation

```
Priority 3 : 282 / 377
Priority 2 : 362 / 419
Priority 1 : 287 / 370

Total number of strips : 1166
Number of strips observed : 931
Number of strips observed during the day : 623
Number of strips observed during the night : 308
Cloud cover < 50% : 434
Cloud cover >= 50% : 497
Mean cloud cover : 0.507
Mean acquisition angle : 24.8 degrees
Number of strips cancelled : 23

Number of images produced (from either the visible or the infrared focal plan) : 1554
Number of images downloaded : 1523 (98.0%)

Plan quality = (99.5,114.552,89.48)
```

Figure 4: Top level interface of the PLANET tool when planning is complete.

them, 377 of priority 3 (the highest), 419 of priority 2, and 370 of priority 1 (the smallest); meteorological forecast built from climatological data. On this instance, planning takes 236 seconds (about 4 minutes), using a 3Ghz Intel processor with 2.5Go of RAM, running under Linux. In the resulting plan, 906 (78%) observations are performed and downloaded, 16 (1%) are performed, but not downloaded, and 244 (21%) not performed at all. Among the observations of priority 3, 280 (74%) are performed. Results are 367 (88%) for priority 2 and 275 (74%) for priority 1.

In order to evaluate the four replanning modes, we considered a scenario where 10 urgent requests of priority 3 (the highest) arrive some minutes before uploading the daily plan. Such a scenario is one of the most stressing for replanning because planning must be performed again over the whole one-day planning horizon. Following such a scenario, we built three replanning instances of increasing difficulty (urgent requests either geographically spread, concentrated on already overloaded areas . . .). Relative efficiency of each mode in terms of quality, stability, and computing time depends on the instance type. Running these four replanning modes in parallel would be an option. Another option would be to run them in sequence. For that, the order according to which modes are called could be determined for each replanning instance by performing a quick analysis of the setting.

Conclusion

We built a planning algorithm which (i) is able to handle all the complex physical constraints (in particular those related to attitude trajectory), (ii) guarantees the production of a plan that may be not optimal, but is really executable thanks to constraint checking, and (iii) is able to produce in some minutes, over a one-day planning horizon, a plan with hundreds of observations and downloads, which covers satellite attitude trajectory as well as observation, data download, satellite pointing, and instrument activations.

We adapted the algorithm to run in a repair mode, taking into account urgent observation requests: modification of, first, the optimization criterion and, then, request priorities and weights in heuristics (in order to favour plan stability).

Algorithms were implemented in the PLANET tool which allows planning and replanning to be performed and produced plans to be visualized in the form of timelines.

References

[Beaumet, Verfaillie, and Charmeau 2011] Beaumet, G.; Verfaillie, G.; and Charmeau, M. 2011. Feasibility of Autonomous Decision Making on board an Agile Earth-observing Satellite. *Computational Intelligence* 27(1):123–139.

[Chien et al. 2010] Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-based Space Operations Scheduling with External Constraints. In *Proc. of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*.

[Fox et al. 2006] Fox, M.; Gereveni, A.; Long, D.; and Serina, I. 2006. Plan Stability: Replanning versus Plan Repair. In *Proc. of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*.

[Lemaître et al. 2002] Lemaître, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6:367–381.

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

César Guzmán
Universidad Politécnica de Valencia
cguzman@dsic.upv.es

Vidal Alcázar
Universidad Carlos III de Madrid
valcazar@inf.uc3m.es

David Prior
Universidad de Granada
dprior@decsai.ugr.es

Eva Onaindía
Universidad Politécnica de Valencia
onaindia@dsic.upv.es

Daniel Borrajo
Universidad Carlos III de Madrid
dborrajo@ia.uc3m.es

Juan Fdez-Olivares
Universidad de Granada, Granada
faro@decsai.ugr.es

Introduction

In this work, we present our ongoing effort on building a domain-independent software platform that integrates basic capabilities for planning, execution, monitoring, re-planning and learning. We name it PELEA after Planning, Execution and LEarning Architecture. The goal is two-fold: first, to provide software engineers a tool that can be used off-the-shelf to easily build planning applications, supporting a rapid prototyping life-cycle; and second to provide planning practitioners a tool that can be highly configured and in which new components replacing the ones that are already integrated can be easily added. Regarding the first goal, the platform currently includes state-of-the-art components for performing a wide range of (meta-)planning tasks, such as: planning (using several paradigms), controlled execution, monitoring of correct plan execution, re-planning when needed, learning of control knowledge, or low-level planning. Ultimately the user could use the tool as-is by giving as input a domain and problem descriptions. Regarding the second goal, it can serve as a benchmark platform for comparing different techniques under the same conditions. For example, a planning expert might want to try out a new re-planning technique on a robot simulator without the need to generate a complete planning-execution-monitoring-replanning architecture. We are currently interfacing the platform with known simulators (videogames and robotic platforms) as well as developing new ones for specific domains (logistics) and even a domain-independent temporal stochastic simulator. We are using this first prototype to develop some applications, such as a robotic system controlled by classical planning and a logistics transportation system.

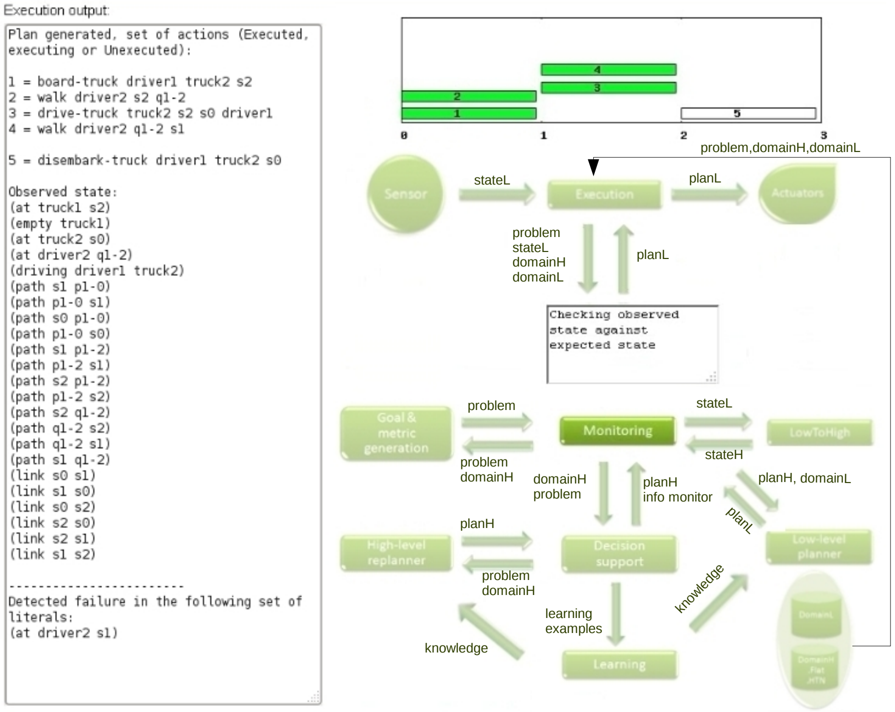
We are building on our combined previous experience on developing different kinds of applications, ranging from fire extinction (Fdez-Olivares et al. 2006), logistics (Flórez et al. 2011), satellites maintenance operations (Rodríguez-Moreno, Borrajo, and Meziat 2004), education (Garrido et al. In Press), tourism (Castillo et al. 2008), or data mining (Fernández et al. In Press), among many others. In all these cases, the process of developing the final application is an "ad-hoc" manual process that requires expertise and techniques on at least two fronts: domain and problem modeling; and selection and configuration of planning systems, together with the implementation of execution controllers, monitoring tools and re-planning techniques, as well as the optional use of learning components. There has been some work on the first task based on powerful modeling tools such as ITSIMPLE (Vaquero et al. 2009). ITSIMPLE allows defining different kinds of planning models, as well as running diverse planners to generate solutions. However, it does not support further execution, monitoring and re-planning of those plans. We propose in this paper a tool that automates those steps.

There has also been previous work that defines generic architectures used for different purposes. Examples can be found in space and robotics applications with platforms like Mapgen (Ai-Chang et al. 2004), APSI (Cesta et al. 2009), PRS (Georgeff and Lansky 1987), or IxTeT (Ghallab and Laruelle 1994). Usually these platforms have been designed for particular planning 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). The goal of the PELEA project is to build a component-based architecture able to perform planning, execution, monitoring and learning in an integrated way, in the context of PDDL-based and HTN-based planning and suitable for a wide range of planning problems.

Next, we define the architecture and its component modules. The architecture allows planning engineers to easily generate new applications that integrate all planning and execution capabilities by reusing and modifying the components. A second scientific advantage of PELEA is to allow researchers or practitioners to compare techniques related to that functionality. We provide a set of tools that implement different techniques for each module, so that users can choose among those. The paper describes the on-going work on this architecture.

Overview of PELEA Architecture

PELEA architecture includes components that allow the applications to dynamically integrate planning, execution, monitoring, replanning and learning techniques. In general, there are 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 lev-



Execution output:

```
Plan generated, set of actions (Executed,
executing or Unexecuted):

1 = board-truck driver1 truck2 s2
2 = walk driver2 s2 q1-2
3 = drive-truck truck2 s2 s0 driver1
4 = walk driver2 q1-2 s1

5 = disembark-truck driver1 truck2 s0

Observed state:
(at truck1 s2)
(empty truck1)
(at truck2 s0)
(at driver2 q1-2)
(driving driver1 truck2)
(path s1 p1-0)
(path p1-0 s1)
(path s0 p1-0)
(path p1-0 s0)
(path s1 p1-2)
(path p1-2 s1)
(path s2 p1-2)
(path p1-2 s2)
(path s2 q1-2)
(path q1-2 s2)
(path q1-2 s1)
(path s1 q1-2)
(link s0 s1)
(link s1 s0)
(link s0 s2)
(link s2 s0)
(link s2 s1)
(link s1 s2)

------------------------
Detected failure in the following set of
literals:
(at driver2 s1)
```

```
Checking observed
state against
expected state
```

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.

els 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. It would be possible to add additional levels to allow developers for a more hierarchical decision process. However, we consider that the sole distinction between high and low level is enough to tackle most problems, as has been shown in many robotics applications. Figure 1 shows a screenshot of PELEA's web interface and the current version of the architecture along with the integration of the modules. Even if we did not provide the explicit APIs, all modules in the architecture have access to either the high-level and low-level domain. We will describe in more detail later on the inputs and outputs of each component.

As we can see, PELEA is composed of eight modules that exchange a set of Knowledge Items (KI) during the reasoning and execution steps. The main KIs that we have used are (the modules also exchange the information related to the parameters that configure how each module works[1]):

- stateL: low-level state composed of the sensory information
- stateH: high-level state, translated from stateL as an aggregation or a generalization of low level information
- goals (problem): the set of high-level goals to be achieved by the architecture
- metrics (problem): the metrics that will be used in the high-level planning process
- planH: set of high level plans. Each high level plan is a set of actions resulting from the high-level planning process. The actions of these plans 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)
- planL: set of low level plans. Each low level plan is again an set of actions resulting from the low-level planning process. These actions should be operational, that is directly executable in the environment
- domainH: definition of actions for high-level planning
- domainL: definition of behaviors (skills) for low-level planning
- learning examples: to be used by the learning component to acquire knowledge for future planning episodes, either in the form of heuristics, domain models, or knowledge on the problem specification
- heuristics: in different forms (control rules, policies, cases, macro-actions, etc.) allow the planners to improve their efficiency in solving future planning episodes

[1] For instance, which planner to execute.

- info monitor: meta knowledge on the plan that helps to perform the monitoring (as, for instance, the generation time of a literal)

The PELEA architecture is controlled by a module, called Top-level control, which coordinates the execution and interaction of the Execution and Monitoring modules. As said above, PELEA architecture uses a two-level knowledge approach. The high-level knowledge describes general information, actions in terms of its preconditions and effects, and typically represents an abstraction of the real problem.

The high-level knowledge descriptions are rarely directly executable, if ever, they must be complemented by the low-level knowledge, which describes the more basic actions in the simulated world, and it is typically concerned with specific rather than general functions, and how they operate. The low-level knowledge is read from the environment through the sensors placed in the Execution module. The environment is either a hardware device, a software application, a software simulator, or a user. An example of low-level knowledge would be "the coordinates of a robot" or "degrees of motion of a robot arm". In PELEA, it is not necessary to work at the two knowledge levels. For instance, 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 needed. A more detailed description of the operation of the architecture of PELEA can be seen in (Alcázar et al. 2010). In the following, the life-cycle of the architecture is described.

Execution Module. The starting point of the architecture is the Execution module, which is initialized by the Top-level control, receiving a high-level and low-level domain, and a problem, composed of an initial state, a set of goals to achieve, a set of objects, and, optionally, a metric. The Execution is initialized with the domain and the problem, which in turn initializes the objects and their positions in the environment. The Execution keeps only the static part of the initial state, given that the dynamic part, called *stateL* (low-level state), will come from the environment through the sensors.

Monitoring Module. *stateL*, the problem and the domain are sent by the Top-level control to the Monitoring module to obtain a low-level plan (*planL*). The actions in *planL* are executed one by one by the Execution module (as can be seen in the Figure 1). As commented above, the modules LowToHigh and Low-level planner are only used in case the domain is modeled at the high and low levels. Otherwise, the Monitoring calls directly the Decision Support to obtain a high-level plan (*planH*). On the other hand, the module Goals&Metric Generation is invoked in case the problem goals or the metric change dynamically along the plan execution. Once the Monitoring module receives the necessary knowledge (state, problem and domain), it starts the monitoring process. The first step of the plan monitoring is to check whether the problem goals have already been achieved (*goalsL* and *goalsH* in case we are dealing with the two processes). If so, the plan execution finishes; otherwise, the Monitor begins with the first iteration of the plan monitoring.

Decision Support Module. At the first iteration of the algorithm, there is no plan to monitor yet, so the Monitoring calls the Decision Support, which obtains a valid plan that achieves the goals from the current observed state through the High-level replanner. This latter module receives a problem and a high-level domain (*domainH*), and generates a high-level plan (*planH*). *planH* is sent back to the Decision Support module, which computes the variables to be monitored and keeps this information in the parameter *info monitor*. Both *planH* and *info monitor* are sent by the Decision Support to the Monitoring.

Low-level Planner. The Monitoring module, with the help of the Low-level planner module, generates a set of executable low-level actions (*planL*), if this is the case. If the Low-level planner module is not being used, the Monitoring assumes that the high-level actions in *planH* are executable, and they are sent to the Execution module, which executes the actions one by one. Then, it senses the dynamic part of the state from the environment. The Monitoring receives the information from the observed state (*stateL*) after the execution of an action, and verifies the information in *stateL* against the parameter *info monitor*. If the values of all the checked variables are within the value range specified in *info monitor*, the Monitoring continues with the plan execution.

Replanning/Plan Repair. Otherwise, if a discrepancy between the expected and the observed state (*stateL*) is encountered, for instance, in the Figure 1 the Monitoring has detected a discrepancy in the literal *at driver2 s1*, which means that the action *walk driver2* has failed in the execution, the anomaly is reported by the Monitoring module to the Decision Support, which determines whether the discrepancy is relevant to the plan execution or not. That is, whether the plan is still valid to achieve the goals from the current observed state. At this point, the low-level planner can also be invoked to find the most immediate actions for a rapid intervention -if reactivity is needed- since this module typically stores predefined behaviours or courses of actions for reaching a situation. In case the Decision Support finds the anomaly entails a plan failure, and so the plan is no longer executable, it will take a decision about whether applying a plan repair, or replanning through the High-level replanner, thus starting a new iteration of the algorithm. Particularly, the Decision Support decides by an Anytime Plan-Adaptation approach (Garrido, Guzman, and Onainda 2010) whether it is worth repairing the plan, in which case it fixes *planH* and makes it executable again, or, it would be better to replan, in which case it requests a new plan to the High-level replanner module. In case that the discrepancy is not relevant to the plan validity, the Decision Support resumes the execution of *planH* by sending back the remaining and the new parameter *info monitor* to the Monitoring module, which in turn sends the next action to the Execution.

Whilst no discrepancies are found in the observed state, the two modules that are continuously interacting are the Monitoring and the Execution. The Monitoring not only checks for discrepancies but also if the problem goals (*goalsL* and *goalsH*) are already satisfied in the current state. In that case, the overall process is finished.

Currently, PELEA integrates, among others, with the following environments: Physical robot PIONEER 3DX through Player (robot independent platform for controlling robots of various kinds); Temporal probabilistic simulator, developed within the project that allows users to define temporal probabilistic domains, in the spirit of MDP-Sim (Younes and Littman 2004), for which we also have an API; Virtual Robot Simulator (VRS[2]) that is a freeware software suite for robotics applications; Alive (Fernández et al. 2008), an open platform for developing social and emotion oriented applications; and TIMI (Florez et al. 2010), a planning tool for real logistic problems.

Conclusions

In this paper, we have presented the ongoing work on building an architecture, PELEA, that integrates planning related processes, such as sensing, planning, execution, monitoring, replanning and learning. It is conceived as a flexible and modular architecture that can accommodate state-of-the-art techniques that are currently used in the whole 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, parametrize 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.

Acknowledgements

This work has been partially supported by the Spanish MICINN project TIN2008-06701-C03.

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.; Milla, G.; Prior, D.; Borrajo, D.; Castillo, L.; and Onaindía, E. 2010. Pelea: Planning, learning and execution architecture. In *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodríguez, A.; Fernández, S.; Arias, J. D.; and Borrajo, D. 2008. SAMAP. A user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(2):1318–1332. ISSN: 0957-4174.

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 21^{st} Innovative Applications of Artificial Intelligence Conference, Pasadena, CA, USA*.

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.

Fernández, S.; Asensio, J.; Jiménez, M.; and Borrajo, D. 2008. A social and emotional model for obtaining believable emergent behavior. In Traverso, P., and Pistore, M., eds., *Artificial Intelligence: Methodology, Systems, and Applications*, volume 5253/2008 of *Lecture Notes in Computer Science*, 395–399. Varna, Bulgaria: Springer Verlag.

Fernández, S.; de la Rosa, T.; Fernández, F.; Suárez, R.; Ortiz, J.; Borrajo, D.; and Manzano, D. In Press. Using automated planning for improving data mining processes. *Knowledge Engineering Review Journal*.

Florez, J. E.; García, J.; Álvaro Torralba; Linares, C.; Ángel Garcia-Olaya; and Borrajo, D. 2010. Timiplan: An application to solve multimodal transportation problems. In Steve Chien, G. C., and Yorke-Smith, N., eds., *Proceedings of the 2010 Scheduling and Planning Applications woRKshop (SPARK'10)*, 36–42.

Flórez, J. E.; Álvaro Torralba; García, J.; López, C. L.; Ángel García-Olaya; and Borrajo, D. 2011. Planning multi-modal transportation problems. In *Proceedings of ICAPS'11*. Freiburg (Germany): AAAI Press.

Garrido, A.; Onaindía, E.; Morales, L.; Castillo, L.; Fernández, S.; and Borrajo, D. In Press. On the automatic compilation of e-learning models to planning. *Knowledge Engineering Review Journal*.

Garrido, A.; Guzman, C.; and Onainda, E. 2010. Anytime plan-adaptation for continuous planning. In *Proceedings of the 28th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG'10)*.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proceedings of AAAI-87 Sixth National Conference on Artificial Intelligence*, 677–68.

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

Rodríguez-Moreno, M. D.; Borrajo, D.; and Meziat, D. 2004. An AI planning-based tool for scheduling satellite nominal operations. *AI Magazine* 25(4):9–27.

Vaquero, T.; Silva, J.; Ferreira, M.; Tonidandel, F.; and Beck, C. 2009. From requirements and analysis to pddl in itsimple3.0. In *Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling*.

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, School Of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

[2] http://robotica.isa.upv.es/virtualrobot/

Dora, a Robot Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Object Search*

Marc Hanheide, Charles Gretton
University of Birmingham
Birmingham, UK

Moritz Göbelbecker
Albert-Ludwigs-Universität
Freiburg, Germany

Abstract

Dora, the robot, is trying to find object in its environment. Instead of just exhaustively searching everywhere, Dora is equipped with probabilistic reasoning, representations, and planning to exploit uncertain common-sense knowledge, such as that cornflakes are usually found in kitchens, while also accounting for the uncertainty of sensing in the real-world. Dora demonstrates how to combine task and observation planning in the presence of uncertainty by autonomously switching between contingent and sequential planning sessions. The demonstration emphasises the benefit of employing a robot with common-sense knowledge and the benefit of the switching planner.

Introduction

With Dora, we are presenting results of our efforts to build a robot capable of performing tasks on demand in dynamic real-world environments. With this paper and demonstration we explicitly address the challenge to perform task and observation planning under uncertainty in pursuit of current robot goals by presenting a new planning approach to reason with new representations of space. For Dora we integrate probabilistic models of background conceptual knowledge, and the visual appearance of objects and of room categories, to solve an object search task. These models are used to create and maintain a probability distribution over possible states with respect to the spatial structure, the categories of objects and rooms, and their relations to each other. Dora, as presented in this paper, is a successor of a previous system (Hawes et al. 2011) that did not make use of probabilistic representations and featured only a classical, sequential planner (Helmert 2006) to achieve exploration and categorisation of rooms.

Related Work

Probabilistic representations are employed for many localised functions in robots operating in the real world. For example, Thrun et al. (2000) use such representations in most of their system's individual components, but their robot behaviour is generated using a reactive controller rather than a domain-independent planner as here.

A number of recent integrated robotic systems incorporate a high-level *continual planning* and execution monitoring subsystem (Talamadupula et al. 2010; Kraft et al. 2008). For the purpose of planning, sensing is modelled deterministically, and beliefs about the underlying state are modelled qualitatively. We are not aware of any robot system that features both a unifying probabilistic representation, and a domain-independent planner which is able to reason quickly over that unified decision-theoretic model to generate behaviour.

Object search with mobile robots has been studied for almost 20 years (Shubina and Tsotsos 2010), yet no previous system reasons with uncertain conceptual knowledge about room and object categories. Instead, most dedicated systems treat the problem as a geometric one. Closest to our approach is the work by Sjöö et al. (2010) who used common-sense knowledge encoded into a rule-based ontology to inform a deterministic planner which previously categorised room to search for a particular object. Bouguerra, Karlsson, and Saffiotti (2007) extended this approach to treat some of the conceptual knowledge as uncertain, although restricted to the number of occurrences of object types in rooms. Vasudevan and Siegwart (2008) went beyond this to perform room categorisation through Bayesian reasoning about the presence of objects, but did not (as none of these did) include observation models in their reasoning (thus perception was still considered to be deterministic).

The Dora System

In order to perform its object search task Dora is equipped with a camera on a pan-tilt unit, a laser scanner, and one laptop accommodating all the processes. The system architecture itself is an extension of PECAS (Hawes, Brenner, and Sjöö 2009) composed of many components functionally structured into subarchitectures. In general, Dora features speech understanding and dialogue components to receive commands from humans, a goal management subsystem (Hanheide et al. 2010) translating commands into goals for the planning subsystem, and many other utility components whose description goes beyond the scope of this extended abstract. At the core of the system is the *switching planner*. Its role is to deliberate Dora's behaviour to ef-

*The research reported here was performed in the EU FP7 IP "CogX: Cognitive Systems that Self-Understand and Self-Extend" (ICT-215181).

ficient object search, exploiting common-sense, relational, and conceptual knowledge. It operates on the probabilistic belief state defined by the conceptual layer of the spatial representation. This probabilistic representation utilises a chain graph model (Lauritzen and Richardson 2002) for inference and integrates conceptual and instance knowledge as detailed by Hanheide et al. (2011), with the latter continuously being maintained by perceptual processes.

Among these are processes that maintain metric and topological maps (Pronobis et al. 2009). Following an approach described by Hawes et al. (2011) the map is discretised into places and rooms, yielding discrete instances to plan with. Also, we employ a continuously running process recognising properties which are evident of the category of rooms following work by Pronobis et al. (2010). Properties being recognised here are the shape of rooms and their visual appearance. Also, we employ an object detector (Mörwald et al. 2010), pre-trained for a set of 19 objects of interest. The planning system can invoke this object detector as part of a sensing action to get evidence about the existence of an object in the current view of the robot.

The probabilistic relations in the conceptual layer have to be quantified appropriately. For probabilistic relations between instances and concepts these are derived from the sensing processes. For the relations of common-sense and conceptual knowledge we either derive them from training sets or from harvesting information from the web. As will be demonstrated, Dora is capable of exploiting the probabilistic knowledge about the co-occurrence of objects and rooms. This relation was quantified employing a combination of qualitative bootstrapping from the *Open Mind Indoor Common Sense* database[1] and queries to an online image search engine. This offline acquisition process yields conditional probabilities, such as $P(room = \text{kitchen}|object = \text{cereal\_box}) = 0.33$.

The Switching Planner

To generate flexible goal-oriented behaviour our system employs a domain-independent planner. The object search scenario poses several challenges to the planning system: On the one hand, planning and execution monitoring must be lightweight, robust, timely, and should span the lifetime of the robot. Those processes must seamlessly accommodate exogenous events, changing objectives, and the underlying *unpredictability* of the environment. On the other hand, in order to act intelligently the agent must perform computationally expensive reasoning about *contingencies*, and possible revisions of subjective belief according to quantitatively modelled uncertainty in acting and sensing.

In our work we take a concrete step towards addressing the challenges we outlined. We have developed a *switching* domain-independent planning system that operates according to the continual planning paradigm. It uses first-order declarative problem and domain representations, expressed in a novel extension of PPDDL (Younes et al. 2005) called *Decision-Theoretic (DT)PDDL*, for modelling stochastic decision problems that feature partial observability. The system *switches*, in the sense that the underlying planning procedure changes depending on our robot's subjective degrees of belief, and progress in plan execution. When the underlying planner is a deterministic sequential planner, i.e., a *classical* planner, we say planning is in a *sequential* session, and otherwise it is in a *contingent* session. Finally, planning is continual in the usual sense that, whatever the session, plans are adapted and rebuilt online in reaction to changes to the planning model (e.g. when objectives are modified, or when our robot's path is obstructed by a door being closed). By autonomously mixing these two types of sessions our robot is able to be robust and responsive to changes in its environment *and* make appropriate decisions in the face of uncertainty. We will give a brief overview of the approach, a more detailed description can be found in the literature (Göbelbecker, Gretton, and Dearden 2011).

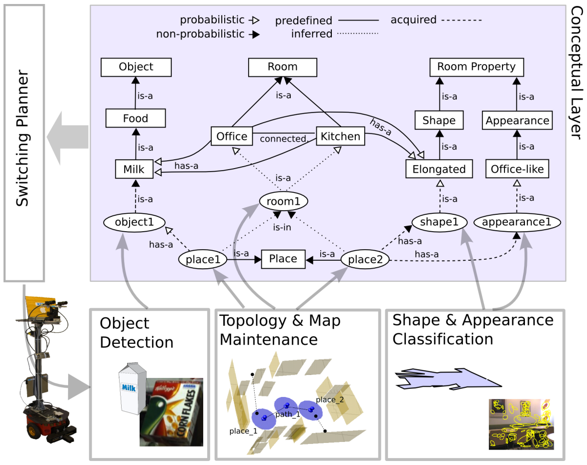


Figure 1: An abstract view of the processes and representations of the system. Sensing processes (at the bottom) discretise and categorise sensor input into instances (shown as ellipses) and acquired relations in conceptual layer. This layer also comprises knowledge about concepts (rectangles) of which only an excerpt in shown. The switching planner reasons upon the state distribution given by the conceptual map.

Sequential Planning

During a sequential session, a rewarding *trace* of a possible execution is computed using a modified version of the cost-optimising satisficing planner *Fast Downward* (Helmert 2006) which trades action costs, goal rewards, and determinacy.

The planning model we use for specifying the sequential planning problems is an extended SAS$^+$formalism (Bäckström and Nebel 1995). In contrast to probabilities in more expressive models like MDPs, actions do not have multiple possible outcomes, they just can succeed with probability $\mathrm{p}(a)$ or transition into a sink state with probability of $1 - \mathrm{p}(a)$. "Real" probabilistic actions can be approximated by creating a separate action for every possible outcome (Yoon, Fern, and Givan 2007). The planner plans according to a cost function c that

[1] http://openmind.hri-us.com/

weights the cost of a plan against its probability. There are several possible choices for how to combine costs and probabilities, we chose a function that resembles the expected reward adjusted to our restricted planning model. With R being a reward constant, we minimise the formula $\mathrm{c}(\pi) = \sum_{a\in\pi} \mathrm{c}(a) + R\left(1 - \prod_{a\in\pi} \mathrm{p}(a)\right)$. For small values of R the planner will prefer cheaper but more unlikely plans, for larger values more expensive plans will be considered.

Assumptions To model uncertain initial states (which are an essential feature of exploration problems), we introduce the concept of *assumptive actions*. The initial state of the planning problem is the set of necessarily true propositions. Assumptive actions are then used to add other, uncertain, propositions (assumptions) to the state. Provided that the plan is optimal, only assumptions that help achieving the goal will be included, preferring ones that are more likely.

If, for example, the initial state contains uncertainty about the category of a room, with $P(\mathrm{cat(room)} = \mathrm{kitchen}) = 0.5$, $P(\mathrm{cat(room)} = \mathrm{office}) = 0.3$ $P(\mathrm{cat(room)} = \mathrm{corridor}) = 0.2$. We would then add the assumptions:

$$
\begin{aligned}
\mathrm{pre}(a_1) =& \mathrm{pre}(a_2) = \mathrm{pre}(a_3) = \{\mathrm{def}_{cat(\mathrm{room})} = \bot\} \\
\mathrm{eff}(a_1) =& \{\mathrm{cat(room)} = \mathrm{kitchen}, \mathrm{def}_{\mathrm{cat(room)}} = \top\} \\
\mathrm{p}(a_1) =& 0.5 \qquad \mathrm{c}(a_1) = 0 \\
\mathrm{eff}(a_2) =& \{\mathrm{cat(room)} = \mathrm{office}, \mathrm{def}_{\mathrm{cat(room)}} = \top\} \\
\mathrm{p}(a_2) =& 0.3 \qquad \mathrm{c}(a_2) = 0 \\
\mathrm{eff}(a_3) =& \{\mathrm{cat(room)} = \mathrm{corridor}, \mathrm{def}_{\mathrm{cat(room)}} = \top\} \\
\mathrm{p}(a_3) =& 0.2 \qquad \mathrm{c}(a_3) = 0
\end{aligned}
$$

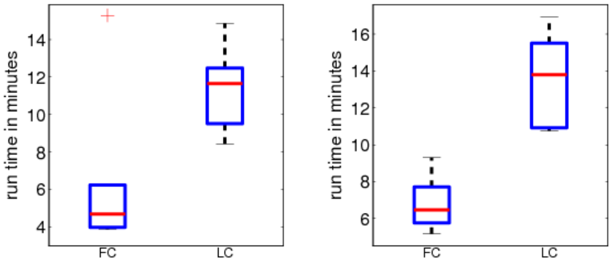
The def-variable makes sure that we cannot make more than one assumption about the same variable.

To utilise background conceptual knowledge, e.g. the probability of finding an object in a certain type of room, we use operators that model the conditional dependencies[2]:

```
(:action object-in-room
   :parameters (?cl - class ?r - room
                ?c - category)
   :probability (P-obj-given-category ?cl ?c)
   :precondition (= (cat ?r) ?c)
   :effect (obj-exists ?cl in ?r))
```

where `(P-obj-given-category ?cl ?c)` are fluents containing the probabilities. Using these operators, we do not have to construct the entire initial state description of the problem explicitely (as we did in the original description of the switching planner).

The system always begins with a sequential session, and once *Fast Downward* produces a trace, plan execution proceeds by applying actions from that trace in sequence until the applicability of the next scheduled action is too uncertain according to a threshold parameter (here, set at 95%). A contingent session then begins which tailors sensory processing to determine whether the assumptions made in the trace hold, or which otherwise acts to achieve the overall objectives.



(a) BHAM: 19 runs. (b) KTH: 10 runs.

Figure 2: Box and whisker diagrams of total runtime to achieve the given task in two environments comparing the ’full’ system (FC) to the ’lesioned’ case (LC).

Contingent Planning

Because decision-theoretic planning in large problems is too slow for our purpose (we seek response times in seconds), contingent sessions plan in an abstract decision process determined by the current trace and underlying belief-state. This abstraction is constructed by first excluding all propositions that are not true of any state in the trace, then adding them back, using as a heuristic the entropy of the trace assumptions conditional on a candidate proposition. Propositions are added, one at a time, until the number of states in the initial belief-state reaches a given threshold (here, 150 states). To the resulting abstract model we also add *disconfirm* and *confirm* actions that the contingent session can schedule in order to judge an atomic assumption in the trace. In the abstract model these actions yield a small reward if the corresponding judgement is true (or small penalty otherwise). Once a judgement action is scheduled for execution the contingent session is terminated, and control is returned to a sequential session.

Experimental Evaluation

In order to test the effectiveness of (i) exploiting default probabilistic knowledge in a conceptual layer of our representation, (ii) the switching planner, and (iii) our implementation of the overall system, we ran two configurations (’full’ and ’lesioned’) of the system in two natural world environments; a residential house in Birmingham (BHAM) and a floor of offices and a kitchen at KTH Stockholm (KTH).

Our evaluation compares the full system with a lesioned system in which the categorisation of visual appearance and shape properties has been disabled, emulating the limited reasoning capabilities available in our previous system (Hawes et al. 2011), where no such evidence was available. The task in all these runs was to find a box of cornflakes. The starting position of the robot was either the living room (in BHAM) or an office (in KTH), i.e. rooms that according to the acquired common-sense knowledge are quite

[2] def-conditions and effects are omitted for clarity

unlikely to contain objects of type cornflakes. This was chosen to showcase the benefit of the probabilistic representation and planning.

Fig. 2 shows the overall runtime to complete the object search task in the lesioned (denoted as 'LC' in the figure) and the full system ('FC') in both environments. What can clearly be seen from the figure is that the full system which can exploit the evidence about the categories of rooms achieves the task significantly faster (Mann-Whitney test $p < 0.01$ for both environments) on average. It benefits from the probabilistic common-sense knowledge that it is quite unlikely to find cornflakes in the room the robot was in and made it decide to first drive to the kitchen to start the search there. On the contrary, in the lesioned case the robot had less information and had to conduct a full exhaustive search. So it started its search in the living room or office, respectively, because the object is as likely to be in this room than in any other. Further details and a more exhaustive analysis of the results are given in (Hanheide et al. 2011).

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS$^+$ planning. *Comp. Intell.* 11(4):625–655.

Bouguerra, A.; Karlsson, L.; and Saffiotti, A. 2007. Handling uncertainty in semantic-knowledge based execution monitoring. In *Proc. Int. Conf. Intelligent Robots and Systems (IROS)*, 437–443.

Göbelbecker, M.; Gretton, C.; and Dearden, R. 2011. A switching planner for combined task and observation planning. In *Twenty-Fifth Conference on Artificial Intelligence (AAAI-11)*.

Hanheide, M.; Hawes, N.; Wyatt, J. L.; Göbelbecker, M.; Brenner, M.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Zender, H.; and Kruijff, G.-J. M. 2010. A Framework for Goal Generation and Management. In *Proceedings of the AAAI Workshop on Goal-Directed Autonomy*.

Hanheide, M.; Hawes, N.; Gretton, C.; Zender, H.; Pronobis, A.; Wyatt, J.; Göbelbecker, M.; and Aydemir, A. 2011. Exploiting Probabilistic Knowledge under Uncertain Sensing for Efficient Robot Behaviour. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI)*. to appear.

Hawes, N.; Hanheide, M.; Hargreaves, J.; Page, B.; Zender, H.; and Jensfelt, P. 2011. Home Alone : Autonomous Extension and Correction of Spatial Representations. In *Proc. Int. Conf. on Robotics and Automation*.

Hawes, N.; Brenner, M.; and Sjöö, K. 2009. Planning as an architectural control mechanism. In *HRI '09: Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, 229–230. New York, NY, USA: ACM.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Research* 26:191–246.

Kraft, D.; Başeski, E.; Popović, M.; Batog, A. M.; Kjær-Nielsen, A.; Krüger, N.; Petrick, R.; Geib, C.; Pugeault, N.; Steedman, M.; Asfour, T.; Dillmann, R.; Kalkan, S.; Wörgötter, F.; Hommel, B.; Detry, R.; and Piater, J. 2008. Exploration and planning in a three-level cognitive architecture. In *CogSys*.

Lauritzen, S. L., and Richardson, T. S. 2002. Chain graph models and their causal interpretations. *J. Roy. Statistical Society, Series B* 64(3):321–348.

Mörwald, T.; Prankl, J.; Richtsfeld, A.; Zillich, M.; and Vincze, M. 2010. BLORT – The Blocks World Robotic Vision Toolbox. In *Proc. ICRA Workshop Best Practice in 3D Perception and Modeling for Mobile Manipulation*.

Pronobis, A.; Sjöö, K.; Aydemir, A.; Bishop, A. N.; and Jensfelt, P. 2009. A framework for robust cognitive spatial mapping. In *Proceedings of the 14th International Conference on Advanced Robotics (ICAR09)*.

Pronobis, A.; Mozos, O. M.; Caputo, B.; and Jensfelt, P. 2010. Multi-modal semantic place classification. *Int. J. Robot. Res.* 29(2-3):298–320.

Shubina, K., and Tsotsos, J. 2010. Visual search for an object in a 3D environment using a mobile robot. *Computer Vision and Image Understanding* 114(5):535–547.

Sjöö, K.; Zender, H.; Jensfelt, P.; Kruijff, G.-J. M.; Pronobis, A.; Hawes, N.; and Brenner, M. 2010. The Explorer system. In Christensen, H. I.; Kruijff, G.-J. M.; and Wyatt, J. L., eds., *Cognitive Systems*. Springer. 395–421.

Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.; and Scheutz, M. 2010. Planning for human-robot teaming in open worlds. *ACM Trans. Intell. Syst. Technol.* 1:14:1–14:24.

Thrun, S.; Beetz, M.; Bennewitz, M.; Burgard, W.; Cremers, A.; Dellaert, F.; Fox, D.; Hähnel, D.; Rosenberg, C.; Roy, N.; Schulte, J.; and Schulz, D. 2000. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research* 19(11):972–999.

Vasudevan, S., and Siegwart, R. 2008. Bayesian space conceptualization and place classification for semantic maps in mobile robotics. *Robot. Auton. Syst.* 56:522–537.

Yoon, S.-W.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *Proc. Int. Conf. Automated Planning and Scheduling (ICAPS)*.

Younes, H. L. S.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res.* 24:851–887.

A Path Planning Algorithm for an AUV Guided with Homotopy Classes Demo Storyboard

Emili Hernandez and **Marc Carreras** and **Pere Ridao**
Department of Computer Engineering
University of Girona
17071 Girona, Spain

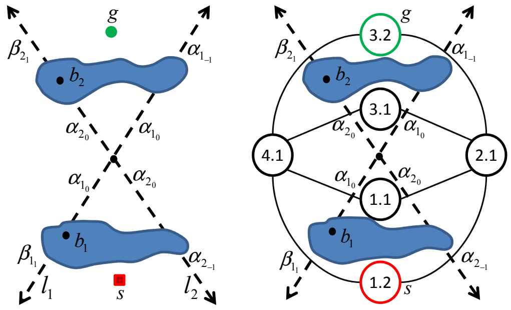
Abstract

This document presents a demo storyboard for an accepted paper in the ICAPS 2011. The paper presents a method that builds a topological environment based on the workspace to compute homotopy classes, which topologically describe how paths go through the obstacles in the workspace. Then, the homotopy classes are sorted according to an heuristic estimation of their lower bound. Only those with smaller lower bound are used to guide a planner based on the Rapidly-exploring Random Tree (RRT), called *Homotopic RRT* (HRRT), to compute the path in the workspace. The demo consist of a short presentation where the theoretical concepts of our paper are explained and an execution of an application that applies the whole method in bitmap scenarios[1].

Introduction

This storyboard presents the practical part of a paper accepted in the ICAPS 2011 (Hernández, Carreras, and Ridao 2011). The paper presents an extension of the method to generate homotopy classes that can be followed in any 2D workspace. Two paths that share the start and the end points belong to the same homotopy class if one can be deformed into the other without encroaching any obstacle. The homotopy classes generated are sorted according their quality given by a lower bound estimator. In order to maximize the number of homotopy classes that can be explored, we propose a path planner based on the RRT, called *Homotopic RRT* (HRRT), to generate paths in the workspace. The HRRT algorithm starts looking for a path in the homotopy class that has a high probability of containing the optimal solution. Our method has been tested in several scenarios and this demo will show up all the execution steps during these tests.

The demo consists of two steps. First, there will be a small presentation to introduce the goal of our research work, the theoretical concepts of homotopy classes and how are they generated in 2D workspaces. Then, there will be an application to run the tests proposed in the paper and see how our method overcomes each step of the process.



a) Reference frame b) Topological graph

Figure 1: Example of a reference frame and its correspondent topological graph in a workspace with two obstacles.

Demo Presentation

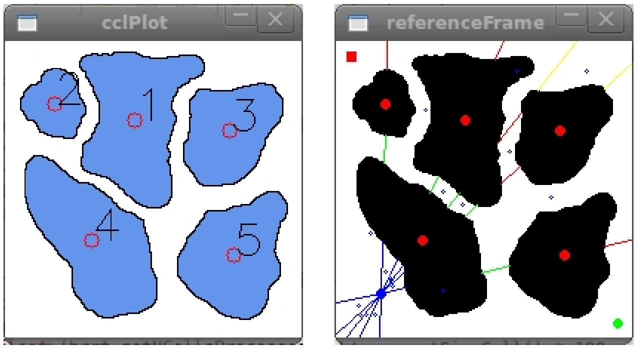
We will briefly present in few slides the theoretical concepts of the paper: the process of turning a metric space into a topological graph (Figure 1), the generation of the homotopy classes and computation of their lower bound, and the fundamentals of the HRRT path planning algorithm. The presentation will also include a slide with the goal of our method: its application to an Autonomous Underwater Vehicle (AUV).

Demo Application

The execution of the demo application will allow the user to see all the steps of the method we propose to compute paths following homotopy classes. The user will be able to select a scenario among a set of bitmaps, to select the start point and goal point through a GUI, and to run the application.

The execution steps are these follows: Given a workspace, our method computes the reference and its topological graph. Both of them are shown in Figure 2 and in Figure 3 to easily understand how a metric environment can be turned into a topological one. Using the topological graph and knowing where the start and goal vertexes are (through the location of the start and goal points in the workspace), a set of homotopy classes will be computed. The computed homotopy classes, shown in Table 1, change according to the scenario and/or the start and goal points locations. Then, the application will compute the lower bound for each ho-

[1] The full paper describing this system appears in the ICAPS-2011 Proceedings at pages 82-89.



a) Scenario b) Reference frame

Figure 2: An environment where the obstacles have been identified and its corresponding reference frame.

Index	Homotopy class	Lower buond (pixels)
1	A5,0 A3,0 A4,0 A1,0 A2,0	295.658
2	A2,1 A1,1 B4,1 A3,1 A5,1	262.768
3	A2,1 A1,1 B4,1 A3,1 B5,2	280.403
4	A2,1 B1,2 B4,2 A3,2 A5,1	373.653
5	A2,1 B1,2 B4,2 A3,2 B5,2	357.347
6	A2,1 B1,2 B4,3 B3,3 A5,1	447.938
7	A2,1 B1,2 B4,3 B3,3 B5,2	386.538
8	B2,2 A1,1 B4,1 A3,1 A5,1	266.443
9	B2,2 A1,1 B4,1 A3,1 B5,2	283.623
10	B2,2 B1,2 B4,2 A3,2 A5,1	320.942
11	B2,2 B1,2 B4,2 A3,2 B5,2	304.636
12	B2,2 B1,2 B4,3 B3,3 A5,1	395.227
13	B2,2 B1,2 B4,3 B3,3 B5,2	333.827

Table 1: Homotopy classes of the scenario with with their index and lower bound. Notice that the α_{k_s} / β_{k_s} notation of the paper is represented as Ak, s / Bk, s in the demo application.

motopy class. The lower bound is used to set up a preference order when computing the homotopy classes path in the workspace. Finally, the user will see the execution of the HRRT algorithm we propose in the paper. This part will show the construction of the tree and the path found (Figure 4).

Acknowledgments

This research was partially supported by the Spanish government under the grant DPI2008-06545-C03-03 and the TRIDENT EU FP7-Project under the grant agreement No: ICT-248497.

References

Hernández, E.; Carreras, M.; and Ridao, P. 2011. A path planning algorithm for an AUV guided with homotopy classes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*.

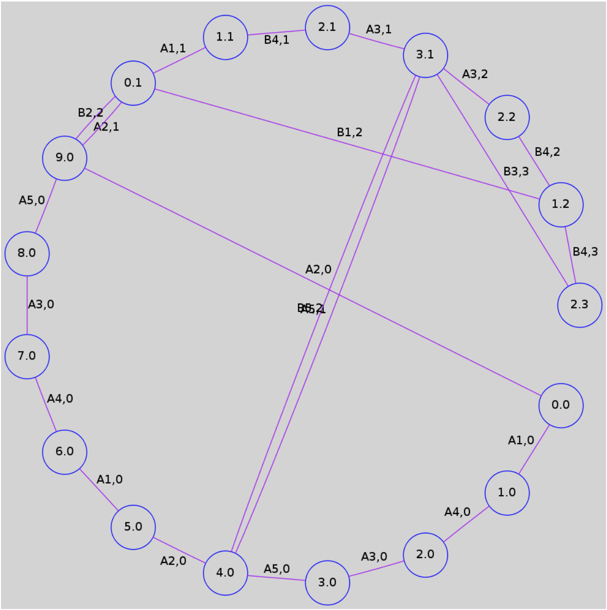
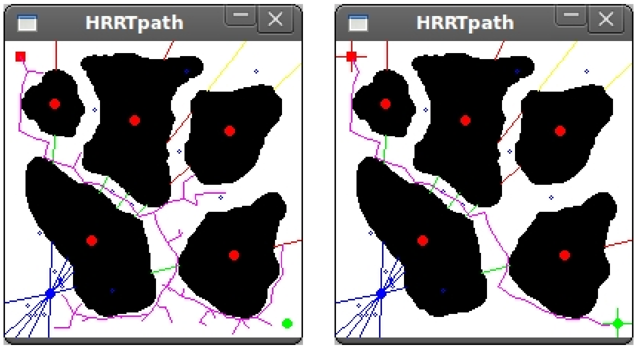


Figure 3: Topological graph of the scenario generated with Graphviz software.



a) HRRT tree b) HRRT path

Figure 4: Tree and path computed with the HRRT algorithm for the homotopy class in Table 1 with the best lower bound (index 2).

The TorchLight Tool: Analyzing Search Topology Without Running Any Search

Jörg Hoffmann
INRIA
Nancy, France
joerg.hoffmann@inria.fr

Abstract

The ignoring delete lists relaxation is of paramount importance for both satisficing and optimal planning. In earlier work (Hoffmann 2005), it was observed that the optimal relaxation heuristic h^+ has amazing qualities in many classical planning benchmarks, in particular pertaining to the complete absence of local minima. The proofs of this are hand-made, raising the question whether such proofs can be lead automatically by domain analysis techniques. The TorchLight tool answers this question in the affirmative.

The tool is based on a connection between causal graph structure and h^+ topology. It distinguishes between *global analysis* and *local analysis*. Global analysis shows the absence of local minima once and for all, for the entire state space of a given planning task. Local analysis determines the percentage of individual sample states not on local minima, thus allowing to make finer distinctions. Finally, *diagnosis* summarizes structural reasons for analysis failure, thus indicating domain aspects that may cause local minima.

Complementing the ICAPS'11 and JAIR papers on TorchLight (Hoffmann 2011b; 2011a), we provide a brief summary of TorchLight's workings and results, and illustrate its functionalities with example output on some IPC benchmarks.

Introduction

The ignoring delete lists relaxation is of paramount importance for both satisficing and optimal planning (e.g., Bonet and Geffner 2001; Hoffmann and Nebel 2001; Richter and Westphal 2010; Helmert and Domshlak 2009). The planners based on it approximate, in a variety of ways, the optimal relaxation heuristic h^+ which itself is **NP**-hard to compute. As was observed in earlier work (Hoffmann 2005), h^+ has strong qualities in many classical planning benchmarks. Figure 1 gives an overview of these results (omitting ADL domains and including the more recent benchmarks Elevators and Transport (without action costs).

The results divide domains into classes along two dimensions. We will herein ignore the horizontal dimension, which pertains to dead ends. The vertical dimension divides the domains into three classes, with respect to the behavior of exit distance, defined as $d - 1$ where d is the distance to a state with strictly smaller h^+ value. In the "easiest" bottom class, there exist constant upper bounds on exit distance from both, states on local minima and states on benches (flat regions). In the figure, the bounds are given in square brackets. For example, in Logistics, the bound for local minima is 0 – meaning that no local minima exist at all – and the bound for benches is 1. In the middle class, a bound exists only for local minima; that bound is 0 (no local minima) for all domains shown. In the "hardest" top class, both local minima and benches may take arbitrarily many steps to escape.

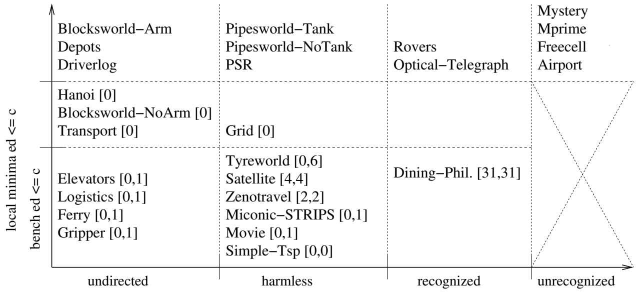


Figure 1: Overview of h^+ topology (Hoffmann 2005).

The proofs underlying Figure 1 are hand-made. For dealing with unseen domains, the question arises whether we can design domain analysis methods leading such proofs automatically. The TorchLight tool answers this question in the affirmative. The key to the analysis is a connection between causal graph structure and h^+ topology. In its most basic form, the connection is this:

If the causal graph is acyclic, and every variable transition is invertible, then there are no local minima under h^+.

The proof of this result works in two steps. Step (A) identifies circumstances under which one can deduce from an optimal relaxed plan for a state s that there exists a monotone exit path, i.e., a path from s to a state s' with $h^+(s') < h^+(s)$ and where all intermediate states s'' on the path have $h^+(s'') = h^+(s)$. Step (B) devises causal graph based sufficient criteria implying that analysis (A) will always succeed. This scheme can be used to prove results stronger than the above, allowing e.g. casual graph cycles arising (only) due to transition "side effects" that are harmless in certain ways.

TorchLight distinguishes between *global analysis* and *local analysis*. Global analysis shows the absence of local minima once and for all, for the entire state space of a given planning task. This is based on step (B) above. Local analysis determines the percentage of individual sample states not on local minima – we refer to this as the *success rate* – thus allowing to make finer distinctions in planning tasks where local minima do exist. To analyze a given sample state s, we feed step (A) with the relaxed plan for s computed by FF's

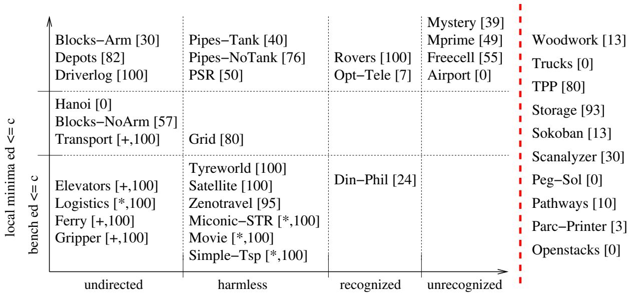


Figure 2: Overview of TorchLight domain analysis results. "*": global analysis always succeeds; "+": local analysis always succeeds if provided an optimal relaxed plan; mean success rates when sampling one state per domain instance.

heuristic function. Since this relaxed plan is not necessarily optimal, this local analysis is approximate: if it succeeds, there is no guarantee that s is indeed not a local minimum.

TorchLight is implemented in C based on FF. Its analysis techniques rely on the finite-domain variable representation of planning. This is obtained from the PDDL input by running Fast Downward's translator (Helmert 2009). That translation is the main bottleneck in TorchLight's runtime performance. Up to 100 sample states, in more than 96% of the 1160 test instances in our experiment, the actual analysis takes at most as much time as the translator.

Figure 2 gives an overview of TorchLight's analysis results. The domains whose h^+ topology is not known are shown separately. For each domain, "*" and "+" indicate domain-specific performance guarantees that we have proved. The numbers give the per-domain average success rates when taking a single sample state per instance. Clearly, "harder" domains tend to have lower success rates.[1]

TorchLight's *diagnosis* summarizes structural reasons for analysis failure, thus indicating domain aspects that may cause local minima. Since the tested criteria are sufficient but not necessary, there is no correctness guarantee. Still, at least for local analysis, the diagnosis can be quite accurate. In Zenotravel, it always correctly identifies fuel consumption as the problem. In Mprime and Mystery, most of the time the same correct diagnosis is returned. In Satellite and Rovers, it always reports the problem to be that switching on an instrument, respectively taking an image, deletes calibration – precisely the only reason why local minima exist here. In Blocksworld-Arm and Freecell, the diagnosis identifies critical resources ("hand-empty" and "have-cellspace").

We next exemplify global analysis, local analysis, and diagnosis, with example runs on IPC benchmarks. We close the paper with a brief discussion of future work.

Global Analysis

Figure 3 gives verbatim output of TorchLight when run on the largest Logistics instance from the 1998 competition (we omit some parts of the output that are not relevant here).

The reader familiar with FF will notice FF's footprint in this output. The run of Fast Downward's translator is indicated by TorchLight near the start of Figure 3. Once translation terminates, TorchLight reads Fast Downward's intermediate output file, and matches the values of the finite-domain variables against FF's grounded facts (this involves a few subtle but uninteresting implementation details).

As visible in Figure 3, TorchLight then builds some basic data structures pertaining to the *support graph* (SG), a simple variant of causal graphs, and the domain transition graphs (DTG) as known from Fast Downward (Helmert 2006). It then sets some basic properties of these structures, for example annotating every individual DTG transition with a flag indicating whether or not the transition is invertible.

Once the basic structures are built and analyzed, TorchLight runs global analysis. This works by enumerating all *global dependency graphs (gDG)*. A global dependency graph is a sub-graph of the support graph that, starting from some goal variable x_0, recursively includes all transitive predecessors of x_0. The gDG is called *successful* if it does not contain any cycles, and satisfies a number of supplementary criteria implying that analysis (A), cf. the above, will succeed. If, and only if, all gDGs are successful – i.e., if as shown here the percentage of successful gDGs is 100% – then it is proved that the state space does not contain any local minima under h^+.[2] Further, each gDG delivers a bound on the exit distance. Maximizing this bound over all gDGs delivers a bound that is valid across the whole state space. In the shown Logistics example, that bound is 1. That same bound would be returned for any Logistics instance, i.e., TorchLight here always finds the exact bound as proved by hand (cf. Figure 1).

Note that the shown instance is huge. FF generates almost a million "action templates", i.e., instantiated actions not yet tested for (relaxed) reachability. This instance size is also reflected in the 9.78 seconds runtime for Fast Downward's translator. By contrast, the actual analysis (i.e., the part of it that we're interested in right now) takes only 0.24 seconds.

As shown in Figure 2, global analysis succeeds in Logistics, Miconic-STRIPS, Movie, and Simple-TSP. In all other domains, however, the fraction of successful gDGs never attains 100%. In these cases, nothing is proved, so those gDGs that are successful may at best serve as an indication of which aspects of the domain are "good-natured".

Local Analysis

Local analysis is run on a set of random sample states. The number R of such states is an input parameter to TorchLight. Each state is sampled by executing $K * h^{\mathrm{FF}}(s_I)$ random actions, where K is another input parameter, s_I is the initial state, and $h^{\mathrm{FF}}(s_I)$ is FF's heuristic value for that state. We start in s_I and keep selecting uniformly one of the applicable actions at each state. The path length factor K is set to 5 in our experiments. We have not played much with this parameter; its value makes a difference mainly in domains containing dead ends (like transportation with non-

[1] In Driverlog and Rovers, deep local minima do exist, but only in awkward situations that don't tend to arise in the IPC instances. Hanoi and Blocksworld-NoArm are not actually easy to solve for FF, and the absence of local minima is due to idiosyncratic reasons.

[2] This is a strictly more general criterion than the one mentioned in the introduction: if the causal graph is acyclic and all transitions are invertible, then all gDGs are successful; but not vice versa.

```
./torchlight -o domains/logistics/domain.pddl -f domains/logistics/p30.pddl

TorchLight: running Fast-Downward translator to generate variables ... done.
TorchLight: creating SG and DTG structures ... done.
TorchLight: static examination of SG and DTG structures ... done.

TorchLight guaranteed global analysis:
No local minima under h+, exit distance bound 1.
Percentage of successful x0/t0 gDGs    : 100.00% (30780 of 30780)

Time spent:    0.14 seconds instantiating 912252 easy, 0 hard action templates
               9.78 seconds in FD translator generating variables
               0.24 seconds in guaranteed global analysis
```

Figure 3: Example run of TorchLight (global analysis) in the Logistics domain.

replenishable fuel), which may not be found if the random walks are too short (we get back to this below).

Given a sample state s, and a relaxed plan $P^+(s)$ for s, local analysis applies step (A) to identify whether or not $P^+(s)$ complies with a special case implying the existence of a monotone exit path from s. If so, we say that s is *successful*. If $P^+(s)$ is optimal, then this analysis is sound, i.e., for successful s an exit path as claimed is guaranteed to exist. In TorchLight, $P^+(s)$ is returned by FF's heuristic function, thus $P^+(s)$ is not necessarily optimal, thus the local analysis is approximate.[3] If s has no relaxed plan at all, then we count the state as unsuccessful.

Upon analyzing all sample states, TorchLight outputs the success rate as well as the min/mean/max exit distance bound identified. Figure 4 gives verbatim output for instances from Transport, Blocksworld-Arm, and Mystery.

Figure 4 (a) shows the output for the largest Transport instance of IPC'08. The "-s 100" in the command line gives the number of sample states (called R herein); the default value is $R = 10$. We see that all sample states are successful, indicating (rightly) the absence of local minima. The largest exit distance bound is 2, however most states have a smaller bound, as indicated by the mean 0.16. Exit distance in Transport relates to the number of vehicle moves needed in order to load/unload the next package. That number can easily be constructed to be large, however apparently this does not tend to happen in the present IPC benchmark instances. As before, we see that Fast Downward's translator constitutes by far the most costly part of the computation. Note, though, that the sampling procedure also takes considerable time (spent in the generation of applicable actions).

In Figure 4 (b), we see a domain, Blocksworld-Arm as run in IPC'00, that does contain local minima under h^+, and where, thus, global analysis is necessarily useless – it can only ever answer "sorry no success". By contrast, approximate local analysis returns interesting information, in terms of the success rate: 25% on one of the largest IPC'00 instance as run here (60 blocks). This indicates (rightly) that there are many states on local minima. Note that, for the 25% successful states, the exit distance is constantly 0, i.e., these are situations where h^+ can be decreased directly due to some simple action that is not intrusive anywhere else.

Consider finally Figure 4 (c), which illustrates the role of dead ends. The Mystery domain of IPC'98, encoding transportation with consumption of non-replenishable fuel, is a classical example of a domain containing such states. Figure 4 (c) shows the run of TorchLight on one of the largest IPC'98 instances (these are not ordered strictly by increasing size). Like in Blocksworld-Arm, the success rate is very low, 34% in this case, rightly indicating the complex nature of the search space surface. However, this time that behavior is mostly due to the presence of dead ends among the sample states, and due to the capability of relaxed planning to recognize these. As visible in the output, 57% of the sample states are recognized to be dead ends. Of the remaining 43 sample states (remember that our total is 100), 34 are successful (and 9 are not). If we sample the states less deeply, by setting K in the random path length $K * h^{\mathrm{FF}}(s_I)$ to $K = 1$ instead of $K = 5$, then only 4 sample states have no relaxed plan, and the success rate skyrockets to 78%.

Diagnosis

There is a variety of information sources in TorchLight that could be used for diagnosis, that is, for the identification of domain features that are good-natured/bad-natured. So far, only a first exploration of this has been made, and only in the context of approximate local analysis. We have implemented a few first-shot methods identifying which operators and variables were involved in the reasons for success/failure of such analyses, in the sample states.

Judging from our current results, the most useful one of these methods reports operators that were "harmful" in the analysis, in that they had "side-effects" preventing them from use in the special case identified by step (A). As an example, consider an operator moving a vehicle, whose intended effect is to change the position of the vehicle, but that has a harmful side effect consuming fuel. The diagnosis reports the name of the operator, along with the name of the predicate affected by the harmful effect. It maintains occurrence counts of these operator-predicate pairs, and weighs these pairs by frequency in order to provide some measure of "importance". Figure 5 gives verbatim output for instances from Mprime, Rovers, and Freecell.

Consider Figure 5 (a). Like Mystery, Mprime encodes transportation with consumption of non-replenishable fuel. In both domains, the available fuel units are associated with locations, rather than with vehicles (the only differ-

[3] TorchLight also implements a version of local analysis guaranteed to be sound. This is based on a localized variant of global dependency graphs. We do not discuss this here since the empirical results are not promising – this local analysis tends to apply only in those domains successfully analyzed by global analysis anyway.

```
./torchlight -o domains/transport/domain.pddl -f domains/transport/p30.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+: 100.00%
Dead-end states:   0.00%
Exit distance bound min:    0, mean:   0.16, max:    2

Time spent:    0.01 seconds instantiating 39304 easy, 0 hard action templates
              11.79 seconds in FD translator generating variables
               3.37 seconds sampling states
               0.52 seconds in approximate local analysis of sample states
```

(a) Transport

```
./torchlight -o domains/blocksworld/domain.pddl -f domains/blocksworld/p61.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+:  25.00%
Dead-end states:   0.00%
Exit distance bound min:    0, mean:   0.00, max:    0

Time spent:    0.00 seconds instantiating 7260 easy, 0 hard action templates
               2.33 seconds in FD translator generating variables
               0.83 seconds sampling states
               2.38 seconds in approximate local analysis of sample states
```

(b) Blocksworld-Arm

```
./torchlight -o domains/mystery/domain.pddl -f domains/mystery/p13.pddl -s 100

TorchLight approximate local analysis of sampled states:
Success and hence no local minima under h+:  34.00%
Dead-end states:   57.00%
Exit distance bound min:    0, mean:   0.00, max:    0

Time spent:    0.03 seconds instantiating 43554 easy, 0 hard action templates
               4.83 seconds in FD translator generating variables
               0.27 seconds sampling states
               0.05 seconds in approximate local analysis of sample states
```

(c) Mystery

Figure 4: Example runs of TorchLight (approximate local analysis) in the Transport, Blocksworld-Arm, and Mystery domains.

ence is that Mprime has an operator allowing to transfer fuel between locations). To discourage planner developers in IPC'98 from analyzing domains and designing domain-specific heuristics, the semantics of both domains was disguised behind meaningless names. One undesirable side effect of this security measure is that the verbatim output in Figure 5 (a) is, also, meaningless. According to Derek Long, more precisely to Long and Fox's (2000) synthesis of generic types, the "feast" operator in Mprime corresponds to a vehicle move, and the "locale" predicate corresponds to the level of available fuel. Thus the analysis in Figure 5 (a) correctly reports the problem to be fuel consumption.[4]

Figure 5 (b) demonstrates a case where the diagnosis identifies a very particular reason for the existence of local minima. Namely, in Rovers, the only reason for their existence is that taking an image has the harmful side-effect of deleting camera calibration. If the same camera is, without a viable alternative, required to take another image, and if re-calibrating the camera involves changing the rover position and thus incurs additional costs, then this side effect may result in a local minimum: the relaxed plan prior to taking the image did not take into account the need to re-calibrate, so after taking the image the relaxed plan length increases. The diagnosis correctly identifies exactly the culprit operator effect. Note, though, that we set $R = 1000$ here. The reason for this is that this kind of awkward situation happens only rarely, so we need a large number of sample states in order to find it. (Even with $R = 1000$, we obtain any diagnosis only in 8 of the 20 IPC'02 Rovers instances.) Note that, with such large R, the runtime advantage of analysis over Fast Downward disappears (in this example at least).

Consider finally Figure 5 (c), in which we demonstrate a case where weighing operator-predicate pairs by frequency is important. Obviously, a major difficulty when playing Freecell is that, when sending a card to a free cell, then the desired effect – making space where the card previously was – is countered by the undesired side-effect of consuming space where the card now is. This is reflected in the diagnosis by the operator-predicate pair "SENDTOFREE (CELLSPACE)". However, there are many other operator-predicate pairs in the diagnosis that are not that sensible, or not sensible at all. For example, "SENDTOFREE (ON)" suggests that, when sending a card to a free cell, the effect

[4] This is not always the case, due to the peculiar encoding of fuel pertaining to locations rather than vehicles. This sometimes tricks the diagnosis into thinking that it's moving away from locations, not fuel consumption, causes the local minima. This never happens in Zenotravel, where fuel pertains to vehicles as one would expect.

```
./torchlight -o domains/mprime/domain.pddl -f domains/mprime/p34.pddl -D

Top weighted non-recovered op/predicate in approximate local analysis:
100.00% of weight -- FEAST (LOCALE)

Time spent:    0.02 seconds instantiating 8964 easy, 0 hard action templates
               1.41 seconds in FD translator generating variables
               0.00 seconds sampling states
               0.00 seconds in approximate local analysis of sample states
```

(a) Mprime

```
./torchlight -o domains/rovers/domain.pddl -f domains/rovers/p19.pddl -D -s 1000

Top weighted non-recovered op/predicate in approximate local analysis:
100.00% of weight -- TAKE_IMAGE (CALIBRATED)

Time spent:    0.02 seconds instantiating 4476 easy, 0 hard action templates
               0.86 seconds in FD translator generating variables
               0.77 seconds sampling states
               0.21 seconds in approximate local analysis of sample states
```

(b) Rovers

```
./torchlight -o domains/freecell/domain.pddl -f domains/freecell/p79.pddl -D

Top weighted non-recovered op/predicate in approximate local analysis:
 58.33% of weight -- SENDTOFREE (CELLSPACE)
 33.33% of weight -- SENDTOFREE (CLEAR)
  8.33% of weight -- SENDTOFREE (ON)

Time spent:    0.05 seconds instantiating 182188 easy, 0 hard action templates
               9.63 seconds in FD translator generating variables
               0.14 seconds sampling states
               0.02 seconds in approximate local analysis of sample states
```

(c) Freecell

Figure 5: Example runs of TorchLight (diagnosis) in the Mprime, Rovers, and Freecell domains. In Mprime, the "feast" operator corresponds to a vehicle move, and the "locale" predicate corresponds to the level of available fuel.

causing trouble is the one removing the card from its previous location. In the example shown, this incorrect diagnosis receives a much smaller weight than the correct one.

Discussion

TorchLight is a new tool whose mission is to analyze search space topology without running any search. What renders this "mission impossible" possible is the observation that causal graphs can be used to characterize rich planning subclasses in which there exist no local minima under h^+.

Apart from furthering our understanding of what makes planning tasks amenable to current heuristic search techniques, such analysis has manifold potential practical uses. In particular, these include: the targeted generation of macro-actions by constructing the identified exit paths; planner performance prediction by machine learning over the generated features; automatic planner/search configuration, even on-line during search since analyzing a single relaxed plan already delivers useful information; automatic problem abstraction by removing (some) harmful effects identified by diagnosis; automatic domain reformulation by using the generated features as reformulation guidance; and PDDL modeling support for end-users by integrating diagnosis as feedback into a modeling environment.

References

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *AI* 129:5–33.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS'09*.

Helmert, M. 2006. The fast downward planning system. *JAIR* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *AI* 173(5-6):503–535.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *JAIR* 14:253–302.

Hoffmann, J. 2005. Where 'ignoring delete lists' works: Local search topology in planning benchmarks. *JAIR* 24:685–758.

Hoffmann, J. 2011a. Analyzing search topology without running any search: On the connection between causal graphs and h^+. *JAIR*.

Hoffmann, J. 2011b. Where ignoring delete lists works, part II: Causal graphs. In *ICAPS'11*, p.98–105.

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *AIPS'00*, 196–205.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *JAIR* 39:127–177.

The “DUKC® Optimiser” Ship Scheduling System

Elena Kelareva*
OMC International / ANU / NICTA
Melbourne, Australia
elena@omc-international.com.au

Abstract

We present an automated ship scheduling system – DUKC® Optimiser – which selects sailing times for a set of cargo ships at a port, so as to maximise total cargo throughput while meeting port operational and safety guidelines, as well as producing schedules that are fair to all companies using the port.

The system has been developed by maritime engineering company OMC International, incorporating elements of the author's PhD research at the Australian National University and NICTA. A prototype of the system has undergone user testing in late 2010, and is planned to undergo further development in order to include additional functionality and incorporate results into a web-based ship management system.

DUKC® Optimiser is the first ship scheduling system that accounts for environmentally-dependent constraints on the times when ships can enter or leave a port. The system uses OMC's existing Dynamic Under-Keel Clearance (DUKC®) software to calculate sailing windows for each ship. The results of the DUKC® calculations are then converted into a Mixed-Integer Programming model, formulated in the MiniZinc modelling language, and solved using the G12 constraint optimisation solver.

Ship Scheduling Background

Ship scheduling deals with assigning sailing times to a fleet of ships, as well as optionally the amount and type of cargo that each ship carries. Ship scheduling is a problem with significant real-world impact, as the majority of the world's international trade is transported by sea, so even a small improvement in schedule efficiency can have significant benefits to industry (Christiansen, Fagerholt, and Ronen 2004).

One consideration in ship scheduling is that most ports have restrictions on the draft of ships that are able to safely enter the port. Draft is the distance between the waterline and the ship's keel, and is a function of the amount of cargo loaded onto the ship. Ships with a deep draft risk running aground when entering or leaving the port, therefore most ports restrict the draft of ships allowed to transit through the port.

In existing ship scheduling algorithms, draft constraints have only been considered in trivial ways, for example, assuming that a given port will always allow ships with a draft of 13 metres or less, and never allow ships with deeper drafts to enter (Fisher and Rosenwein 1989). Other ship scheduling algorithms leave draft constraints entirely up to human schedulers (Fagerholt 2004).

In practice, most ports restrict ship sailing drafts using safety rules that estimate the under-keel clearance (UKC) – the depth of water under a ship's keel. In recent years, OMC International has developed algorithms to accurately calculate under-keel clearance using real-time environmental conditions. OMC's Dynamic Under-Keel Clearance (DUKC®) software allows significantly more cargo to be loaded safely onto each vessel compared to the static UKC rules previously used by most ports, which don't take real-time environmental data into account (OMC 2011). However, ship scheduling has not been able to take advantage of these recent improvements in UKC estimation, due to not considering complex time-varying draft constraints.

In this presentation, we demonstrate the DUKC® Optimiser software, which is the first ship scheduling system that can take environmentally-dependent time-varying draft constraints into account.

Dynamic Under-Keel Clearance

Figure 1 illustrates all aspects of ship motion taken into account by the Dynamic Under-Keel Clearance (DUKC®) software in calculating under-keel clearance. Components of ship motion taken into account by the DUKC® software include:

Draft: the distance from the waterline to the bottom of the ship's keel.

Squat: a phenomenon caused by the Bernoulli effect which causes a ship travelling fast through shallow water to sink deeper into the water than a ship travelling slowly.

Heel: the effect of a ship leaning towards one side, caused by the centripetal force of turning, or the force of wind on the side of the ship.

Wave Response: the motion resulting from the action of waves on the ship. Only the vertical component of this

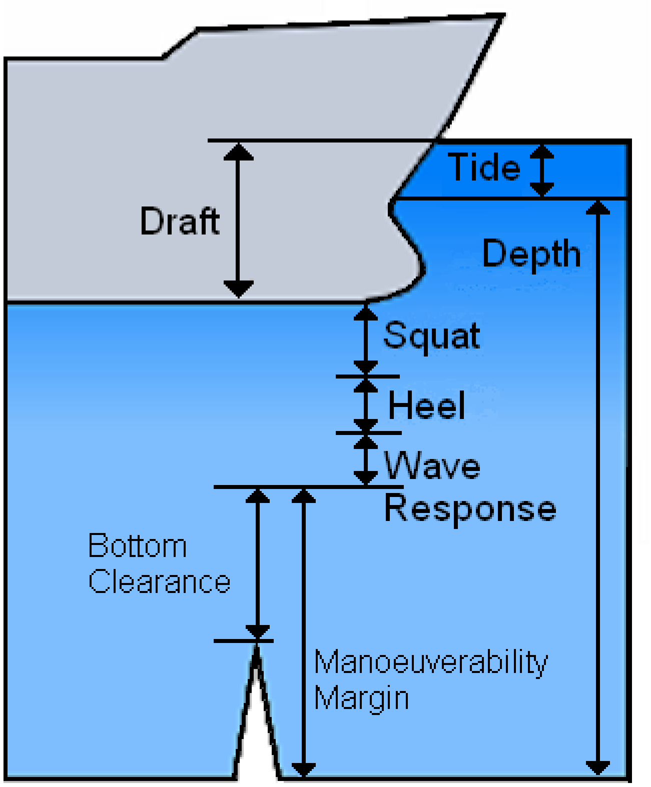


Figure 1: Dynamic Under-Keel Clearance Components

motion affects under-keel clearance.

Under-keel clearance is computed as follows:

UKC = Tide + Depth - Draft - Squat - Heel - Wave Response

The Bottom Clearance and Manoueverability Margin shown in Figure 1 are safety factors that ensure the ship has sufficient distance from the highest points on the channel bottom (Bottom Clearance) and that there is sufficient water around the ship to maintain good manoeuverability (Manoueverability Margin). If the under-keel clearance is below either the Bottom Clearance or Manoueverability Margin safety limits, then the DUKC® software will advise the operator not to sail. However, the DUKC® software only provides navigational advice; the final decision always rests with the ship's pilot or captain.

For a more detailed analysis of Dynamic Under-Keel Clearance methodology, see (O'Brien 2002).

System Architecture

The initial prototype of DUKC® Optimiser is a command line application which uses Microsoft Excel input and output files as a simple "GUI". Excel was used in place of a customised GUI in order to deliver a prototype for user testing as quickly as possible; the system is planned to be incorporated into a web-based under-keel clearance management system in future development. See Figure 3 for a mockup of what a future GUI may look like.

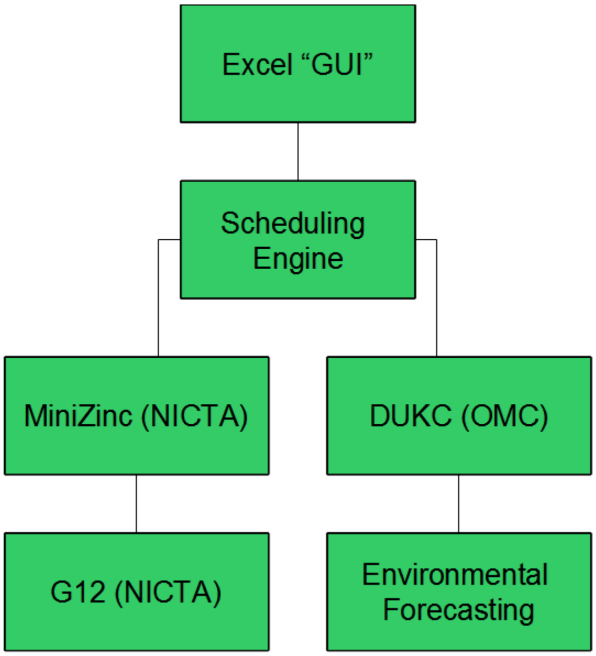


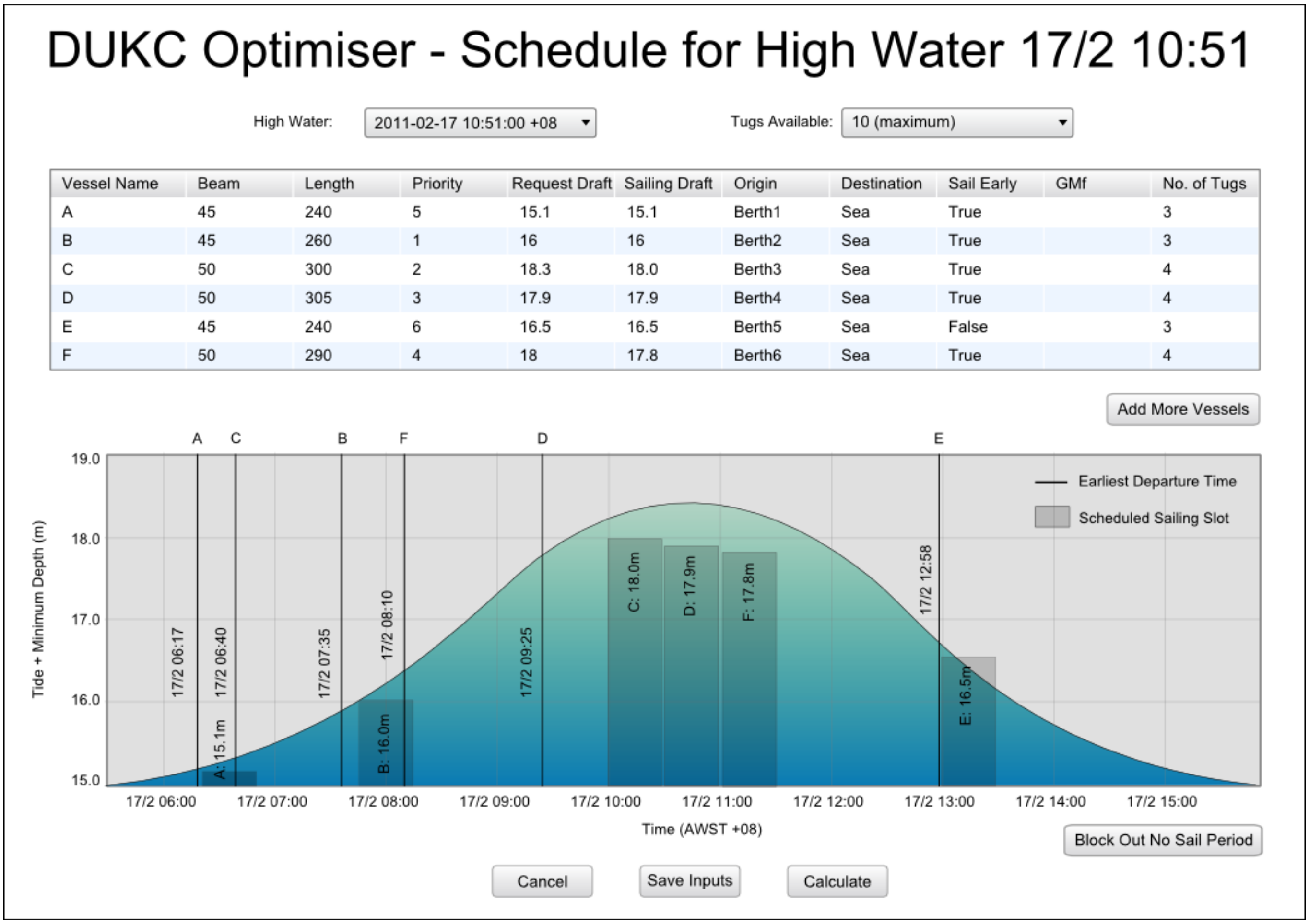
Figure 2: DUKC® Optimiser System Architecture

The scheduler inputs data about each ship into an Excel spreadsheet, which is then read by DUKC® Optimiser and converted into a set of queries to OMC's DUKC® software. The DUKC® software reads real-time environmental forecasts and measurements from databases, and uses these to analyse each ship's motion in response to the predicted tide, wave and current conditions. The results of this analysis is used to calculate under-keel clearance – the amount of water under the ship at each point in the transit – and thus to determine sailing windows for a range of drafts for each ship.

DUKC® Optimiser then converts the user inputs and the results of the DUKC® calculations into a Mixed Integer Programming (MIP) model, implemented in the MiniZinc optimisation language (Nethercote et al. 2007). This model is then solved using the G12 constraint optimisation platform (Stuckey et al. 2005).

MIP Formalisation

The MIP formalisation of the ship scheduling problem includes constraints on the valid range of drafts for each ship, as well as on the sailing draft allowed at each time for each ship by the port's safety rules. Each ship has a minimum and maximum draft range, determined by physical limitations such as the size and shape of the ship, as well as an earliest sailing time, which depends on when the ship finishes loading.



DUKC Optimiser - Schedule for High Water 17/2 10:51

High Water: 2011-02-17 10:51:00 +08

Tugs Available: 10 (maximum)

Vessel Name	Beam	Length	Priority	Request Draft	Sailing Draft	Origin	Destination	Sail Early	GMf	No. of Tugs
A	45	240	5	15.1	15.1	Berth1	Sea	True		3
B	45	260	1	16	16	Berth2	Sea	True		3
C	50	300	2	18.3	18.0	Berth3	Sea	True		4
D	50	305	3	17.9	17.9	Berth4	Sea	True		4
E	45	240	6	16.5	16.5	Berth5	Sea	False		3
F	50	290	4	18	17.8	Berth6	Sea	True		4

Figure 3: Example GUI: Output

The model also incorporates other parameters representing the port's geometry and operational procedures, including locations of significant shallow or narrow points along the channel, travel times between waypoints, and minimum required separation times between ships passing a given waypoint. Constraints on these parameters ensure that ships do not pass through each other, and stay far enough apart to meet the port's safety guidelines.

Objective Function

The objective function for the ship scheduling problem varies per port. Some ports may have an objective function that purely optimises throughput; other ports may need to prioritise fairness to competing clients above optimising the total throughput for the port.

One example of an objective function optimises the total cargo throughput at the port by maximising the sum of the drafts, weighted by the tonnage per centimetre of draft, since the amount of extra cargo allowed by an increase in draft varies depending on the size and shape of the ship.

An alternative objective function, for a port with more complex operating procedures, allows shipping agents to request minimum drafts for each ship, for example to meet contractual obligations. Ships are allocated sailing slots based on priority, which is determined by the port's fairness rules.

Future Development

Future development of the system will include incorporating additional resource constraints to account for tugs, which are used to assist ships entering or leaving the port. Initial user testing conducted in late 2010 found that in some situations, the need to wait for tugs to return from a job constrains the schedule. Therefore tugs need to be incorporated before the system can be used in operation.

Another future development will be to incorporate the system into a web-based under-keel clearance management system, to improve the usability of DUKC® Optimiser for operational use. In the demonstration, we will use mockup screenshots from the future GUI to demonstrate system behaviour, even though the GUI itself has not yet been developed.

A mockup GUI showing input data and an output schedule for a set of six ships sailing on one tide is shown in figure 3.

Acknowledgements

The DUKC® Optimiser system was developed as a project at OMC International. The author would like to acknowledge the contribution of other OMC engineers involved in the development of the DUKC® Optimiser system, particularly Giles Lesser, Gregory Hibbert and Kalvin Ananda. The author would also like to acknowledge the input from her PhD supervisors Philip Kilby, Sylvie Thiébaux and Mark Wallace, and the support of ANU and NICTA at which she is a PhD student. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Christiansen, M.; Fagerholt, K.; and Ronen, D. 2004. Ship routing and scheduling: status and perspectives. *Transportation Science* 38(1):1–18.

Fagerholt, K. 2004. A computer-based decision support system for vessel fleet scheduling - experience and future research. *Decision Support Systems* 37(1):35–47.

Fisher, M., and Rosenwein, M. 1989. An interactive optimization system for bulk-cargo ship scheduling. *Naval Research Logistics* 36(1):27–42.

Nethercote, N.; Stuckey, P.; Becket, R.; Brand, S.; Duck, G.; and Tack, G. 2007. Minizinc: Towards a standard cp modelling language. In Bessière, C., ed., *Principles and Practice of Constraint Programming - CP 2007*, volume 4741 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 529–543.

O'Brien, T. 2002. Experience using dynamic underkeel clearance systems. In *Proceedings of the PIANC 30th International Navigational Congress*, 1793–1804.

2011. OMC International public website. http://www.omc-international.com.au.

Stuckey, P.; de la Banda, M.; Maher, M.; Marriott, K.; Slaney, J.; Somogyi, Z.; Wallace, M.; and Walsh, T. 2005. The g12 project: Mapping solver independent models to efficient solutions. In Gabbrielli, M., and Gupta, G., eds., *Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 9–13.

WindMT: An integrated system for failure detection and maintenance scheduling at wind farms

András Kovács[1], János Csempesz[1], Gábor Erdős[1], Dávid Karnok[1,2], Lőrinc Kemény[1], László Monostori[1,2], Zsolt János Viharos[1]
[1] Fraunhofer Project Center for Production Management and Informatics,
Computer and Automation Research Institute, Hungarian Academy of Sciences, Budapest, Hungary
[2] Department of Manufacturing Science and Technology,
Budapest University of Technology and Economics, Budapest, Hungary

Abstract

The paper introduces a system for supporting the diagnostics and maintenance of wind farms. The work flow in the system covers the detection of failures based on data registered by the SCADA supervisory system of the turbines, the assignment of maintenance tasks to each failure, the scheduling of maintenance tasks on the short-term horizon, and the reporting of maintenance task execution. Finally, various reports are generated from the data collected throughout the above process about the reliability and maintainability of the turbines. In this paper, the primary focus is on maintenance scheduling, using a fine-grained representation of all requirements of the tasks, including resources, spare parts, weather, as well as turbine conditions.

Introduction

Efficient maintenance is crucial for the economic operation of wind energy systems. Wind farm operators must continuously monitor the condition of their turbines, detect if a turbine needs maintenance, and determine the appropriate maintenance action to be performed. Furthermore, the optimal timing and assignment of maintenance tasks to resources must be computed. Maintenance scheduling is not only an important, but also a complex problem. The scheduler must consider the availability of various resources, spare parts, and appropriate skilled personnel, while minimizing the disruptions caused in production.

The paper introduces an integrated system for the diagnosis and maintenance of wind farms. The system, called WindMT, has been developed recently in an EU-funded research project aimed at the improvement of the reliability of wind energy systems, involving wind turbine manufacturers, component suppliers, and research institutes[1]. Below we present the key functionalities of the system, briefly describe the models and algorithms for maintenance scheduling, give some technical details about the developed system, and summarize the screenplay of the planned live demonstration at ICAPS 2011.

Integrated System for Diagnosis and Maintenance

The system performs failure detection and prognosis on quasi-online data from the SCADA supervisory system. In case a failure is detected or prognosed, it assists human experts in initiating the corresponding corrective or predictive maintenance tasks–the available digitized knowledge on mapping failures to tasks is not reliable enough to completely automate this step. Other planned tasks, such as preventive maintenance or retrofitting originate from the ERP system. The developed system schedules the maintenance tasks on a short term horizon so as to minimize production loss due to failures and maintenance. The execution of maintenance tasks is tracked, during which technicians provide valuable detailed information on the nature of the failure, the failed component, and the actual execution of the task, such as its duration and the usage of spare parts. The work flow is summarized in Figure 1.

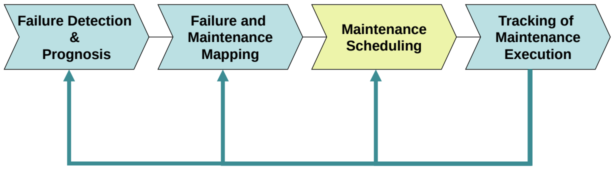

Failure Detection & Prognosis
Failure and Maintenance Mapping
Maintenance Scheduling
Tracking of Maintenance Execution

Figure 1: Key functionalities and the work flow in the system.

The integrated handling of failures, maintenance, and execution makes it possible to feed back the acquired knowledge in order to improve failure detection and prognosis models, the mapping of failures to maintenance tasks, the definition of the maintenance tasks, and through component reliability statistics, even the turbine design. Therefore, the system not only supports the daily operation of the turbines, but it is also a key element for achieving the long-term digitization objectives of the wind farm owner and maintenance service provider companies. Hence, this system represents a significant step towards the practical implementation of the vision stated in (Takata *et al.* 2004) that maintenance should

[1] For further details, check `www.reliawind.eu`.

be a key element of the life cycle management of the products.

Maintenance Scheduling

Modeling the Scheduling Problem

The WindMT system is responsible for the diagnosis and maintenance of multiple wind farms in an area, supervised by the same regional office of the wind farm operator company. There are typically up to 20 farms with at most 300 turbines belonging to a regional office. Since different offices share their resources only in exceptional cases, the scheduling problems of different offices can be modeled as independent.

A detailed maintenance schedule is prepared for a short-term horizon of 3-7 days, on a rolling horizon basis. The schedule is updated every morning, which implies that usually only the tasks of the first day are executed as planned. A key reason for not updating the schedule within the day is that technicians have limited connectivity to the office, therefore notifying them about the modified schedule and acquiring feedback about the execution status in real-time is impossible. Potential changes in the schedule will be managed manually by the experts in the regional office. Furthermore, despite the obvious uncertainties in the scheduling problem, all parameters–including weather conditions and spare part availability–are assumed to be deterministic within the day.

The developed model covers all types of maintenance, including corrective (repairing a failure), preventive (performing periodical maintenance to safeguard from failures) and predictive (condition-based preventive) maintenance, as well as retrofits (upgrading the turbine). From the scheduling perspective, the only difference between these types of maintenance is that corrective maintenance is timely immediately when the task is generated, whereas other, planned maintenance tasks have a target date and an opportunity window, i.e., a time window around the target date in which the task should be executed.

Tasks require different kinds of resources, such as skilled personnel, spare parts, and special equipment. Tasks are executed by technicians, working in fixed teams of two people. A team can execute only one task at a time, and it must finish the task before moving to the location of the next task. However, there are a few extremely long tasks that can be broken into smaller segments or even preempted, i.e., the team may execute another task if circumstances are unsuitable for working on the large task. Traveling from one farm to another one takes a given amount of travel time, whereas the time of travel within the farm can be neglected.

Spare parts are assigned to tasks before scheduling would take place, hence, they imply a release time for the tasks. The availability interval of some special equipment, such as external cranes, may also be limited, hence, the equipment also entails time windows and capacity constraints for the tasks.

An interesting and highly domain specific feature of the scheduling problem is the dependence of maintenance tasks on weather conditions, such as wind speed or temperature. For instance, tasks requiring an outer crane can be executed in calm winds only. Hence, the forecasted weather defines the time periods when certain tasks can be executed. Furthermore, wind speed determines the energy produced by a turbine, and hence, the production loss in periods when the turbine is affected by a failure or stopped for maintenance. More precisely, the production loss incurred by a failure in any period of time can be estimated from the forecasted wind speed, the turbine characteristics, and a loss percent assigned to the failure. Note that multiple tasks/failures can affect the turbine at the same time.

Finally, sets of tasks may interfere beyond competition for common resources as well. Namely, some tasks are executable only when the turbine is (or even multiple turbines are) in a special condition, e.g., with no pressure in the hydraulic system. Pairs of tasks can be executable in parallel only if the turbine conditions are compatible.

Scheduling consists in determining the set of tasks that should be executed within the scheduling horizon and assigning a team and a start time to them, so as to minimize the total production loss, including the due to failures and due to stopping the turbine for maintenance.

Solution Approach

The scheduler uses a combination of mixed-integer programming (MIP) and custom heuristics for solving the above scheduling problem. The core problem is encoded as a MIP using a time-indexed formulation, and it is solved by the default branch and bound algorithm of a commercial MIP solver package. The MIP model is presented in detail in (Kovács *et al.* 2011).

The heuristics perform the pre-processing of the input problem and the post-processing of the schedule computed by the MIP as follows:

- In practice technicians move within a small set of nearby farms only. Often, the bipartite graph of allowed technician-farm assignments contains multiple independent components, which means that the scheduling problem can be decomposed to sub-problems corresponding to zones within the region. In case of very large components, the algorithm forms independent zones by heuristically removing edges from the bipartite graph.
- In case of massively oversubscribed scheduling problems, a heuristic omits the least important tasks. This way, the total volume of tasks in the MIP will exceed the available capacity by at most a given percent, 25% in the experimental settings (travel times and other periods of forced inactivity are ignored here). These heuristics ensure that the size of the core problem faced by the MIP solver will always be tractable.
- Since preemption is allowed for some extremely long tasks, these tasks are partitioned to smaller segments and precedence constraints are posted between the segments.
- In the post-processing step, a heuristic looks for potential improvements of the schedule by re-partitioning and re-assigning the above long tasks.

In computational experiments with the core MIP, the solver constructed exact optimal solutions for problem instances with up to 50 tasks, as well as quality schedules with relative gaps compared to the lower bounds well below 1% for up to 100 tasks. In experiments with the complete solver (including the MIP and the heuristic) the system could handle problems with more than 1000 tasks, which is more than what we expect in a real application.

The WindMT System

WindMT is a Java-based multi-tier application whose presentation layer runs in web browsers on PCs and PDAs (with limited functionality in the latter). The application interfaces with various IT systems of the wind turbine operator and the maintenance service provider companies, including SCADA supervisory system, the enterprise resource planning (ERP) system, the maintenance management system, and a weather forecast service, as shown in the system architecture diagram in Figure 2. The scheduling algorithm is built on the top of the ILOG Cplex 11.2 commercial MIP solver package. Figure 3 shows a screen shot of the system presenting the template for the wind turbine breakdown structure, while Figure 4 displays a screen shot of a Gantt chart representation of a maintenance schedule.

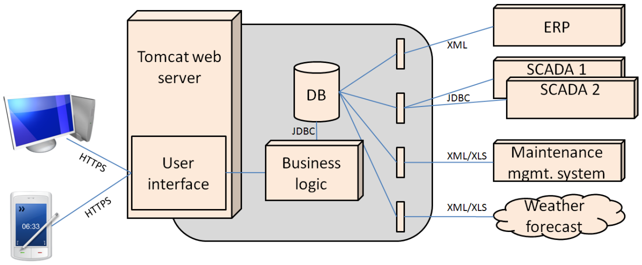
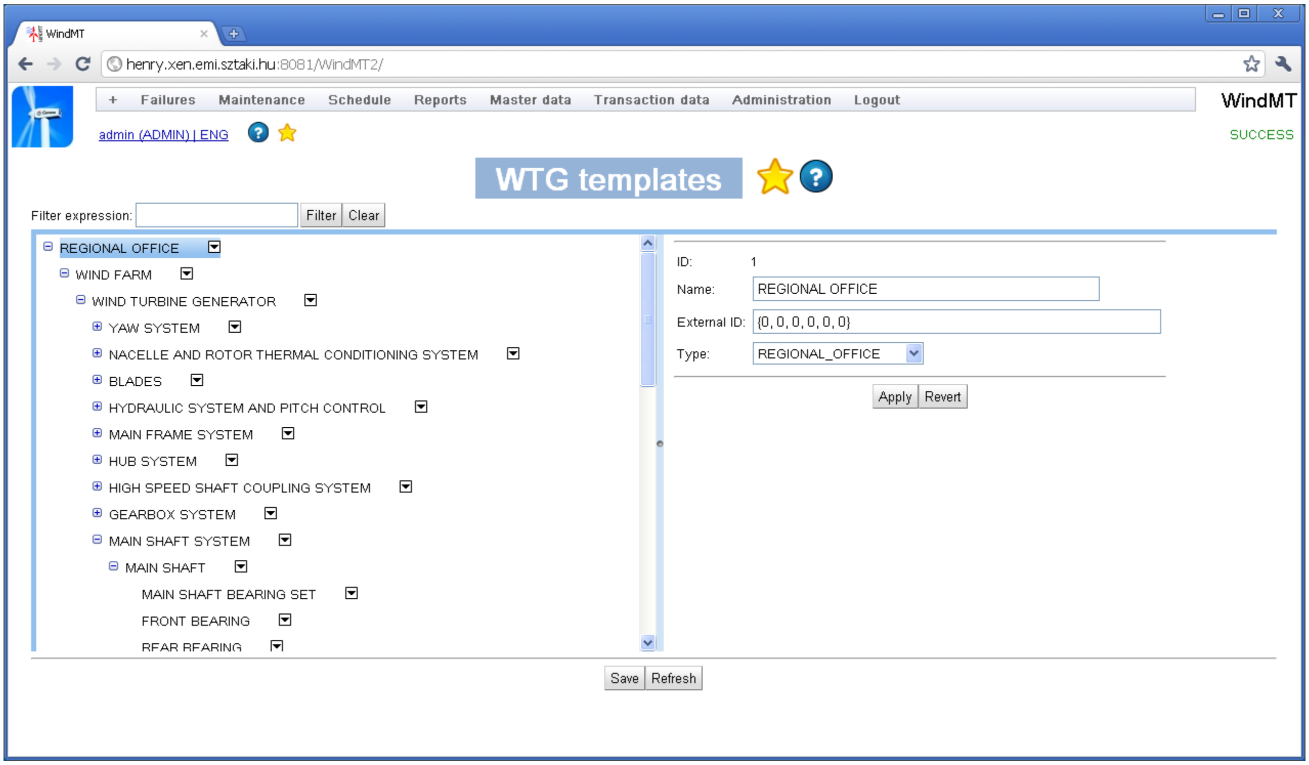


Figure 2: System architecture.

Demonstration

The demonstration planned for ICAPS 2011 covers the complete work flow in the system, starting with the detection of failures, manual and automatic assignment of maintenance tasks to failures, scheduling the maintenance tasks, and finally, execution reporting. In addition, some reports based on the data collected throughout the process will be shown. The demo will be conducted on a small set of manually forged sample data, which is representative but not identical to real-life data of wind turbine manufacturers. Currently,


WindMT
henry.xen.emi.sztaki.hu:8081/WindMT2/
+ Failures Maintenance Schedule Reports Master data Transaction data Administration Logout
WindMT
admin (ADMIN) | ENG
SUCCESS
WTG templates
Filter expression:
Filter
Clear
Apply
Revert
Save
Refresh

⊖ REGIONAL OFFICE
⊖ WIND FARM
⊖ WIND TURBINE GENERATOR
⊕ YAW SYSTEM
⊕ NACELLE AND ROTOR THERMAL CONDITIONING SYSTEM
⊕ BLADES
⊕ HYDRAULIC SYSTEM AND PITCH CONTROL
⊕ MAIN FRAME SYSTEM
⊕ HUB SYSTEM
⊕ HIGH SPEED SHAFT COUPLING SYSTEM
⊕ GEARBOX SYSTEM
⊖ MAIN SHAFT SYSTEM
⊖ MAIN SHAFT
MAIN SHAFT BEARING SET
FRONT BEARING
REAR BEARING

ID: 1
Name: REGIONAL OFFICE
External ID: {0, 0, 0, 0, 0, 0}
Type: REGIONAL_OFFICE

Figure 3: Screenshot of WindMT: tree-structured template of the wind energy system. The tree is rooted in a regional office, responsible for the maintenance of multiple wind farms in an area. Each farm consists of multiple wind turbines. The architecture of a turbine is elaborated in the level of detail that is relevant for statistics about failure occurrences. Individual wind turbines may differ arbitrarily from this template.

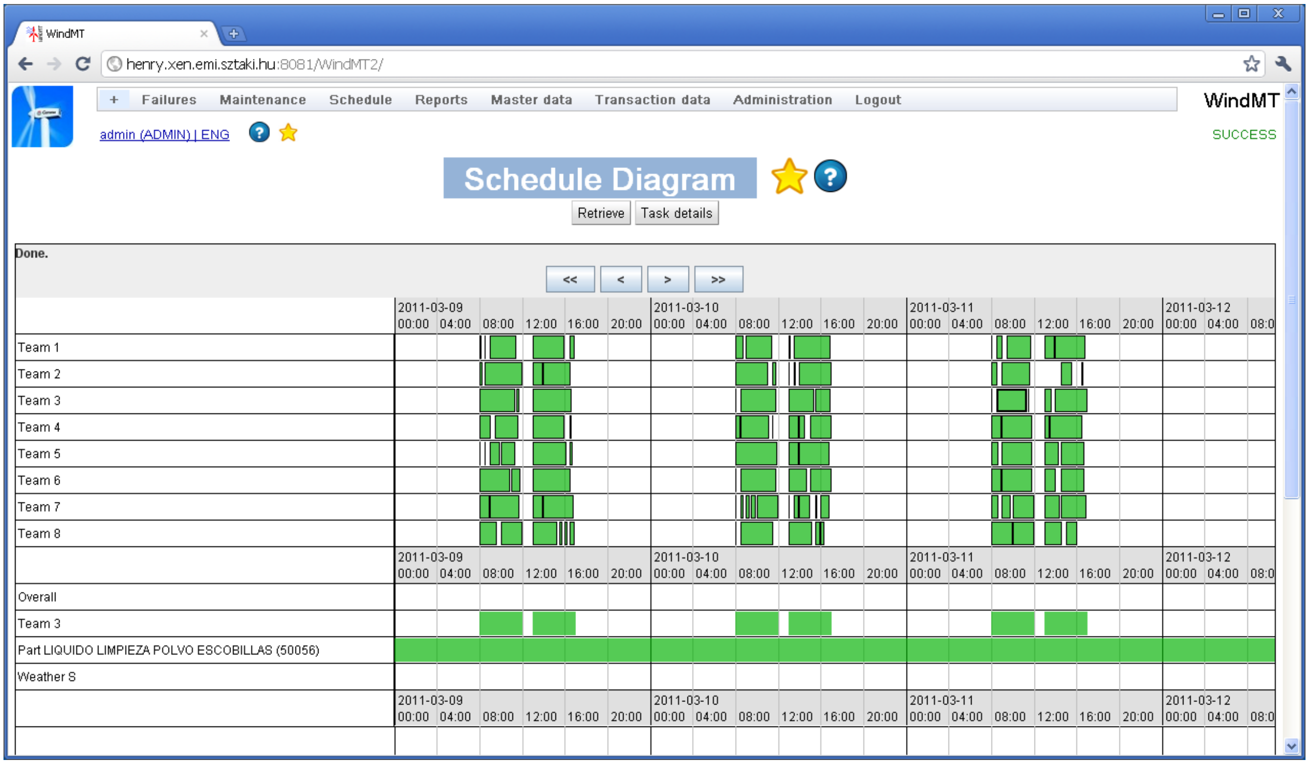


Figure 4: Screenshot of WindMT: maintenance schedule displayed in a Gantt chart. Each of the upper rows contains the tasks to be executed by a specific team in the next three days. The rows in the bottom display the availability of the resources required by the selected task. It is also possible to manually edit the schedule on this screen.

the database of the system contains 100 wind turbines located in 3 farms, ca. 50 failure modes, and master data about 25 different maintenance tasks.

Acknowledgments

The authors acknowledge the support of the EU FP7 grant ReliaWind No. 212966, the Hungarian Scientific Research Fund (OTKA) grant “Production Structures as Complex Adaptive Systems” T-73376, and the National Office for Research and Technology (NKTH) grant “Digital, real-time enterprises and networks”, OMFB-01638/2009 and the bonus grant OMFB-00486/2010.

References

Kovács, A.; Erdős, G.; Viharos, Z.; and Monostori, L. 2011. A system for the detailed scheduling of wind farm maintenance. *CIRP Annals – Manufacturing Technology* to appear.

Takata, S.; Kimura, F.; Van Houten, F. J. A. M.; Westkämper, E.; Shpitalni, M.; Ceglarek, D.; and Lee, J. 2004. Maintenance: Changing role in life cycle management. *CIRP Annals – Manufacturing Technology* 53(2):643–656.

Demonstration of the Emergency Landing Planner

Elif Kurklu[*] **Nicolas Meuleau**[*] **Christian Neukom**[*]
Christian Plaunt **David E. Smith** **Tristan Smith**[†]

Intelligent Systems Division
NASA Ames Research Center
Moffet Field, California 94035-1000
{elif.kurklu, nicolas.f.meuleau, christian.neukom, christian.j.plaunt, david.smith, tristan.b.smith}@nasa.gov

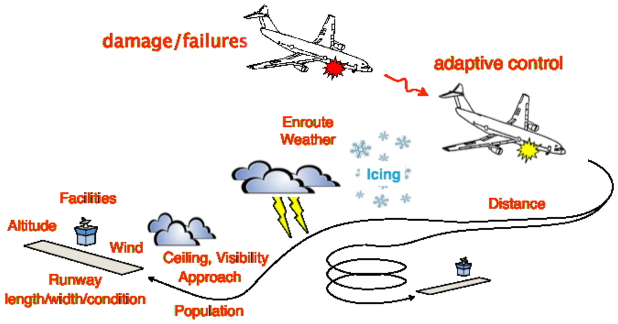


Figure 1: Basic Scenario

Abstract

We describe an Emergency Landing Planner (ELP) designed to assist pilots in choosing the best emergency landing site when damage occurs to an aircraft. In 2010, we integrated the ELP into the cockpit of a 6 DOF full-motion simulator for transport category aircraft, and performed experiments to evaluate the software using crews of professional airline pilots. We briefly describe how the ELP works, and show how it was integrated into the avionics of the simulator.

1. The Scenario

Figure 1 illustrates the type of scenario that the Emergency Landing Planner (ELP) addresses. When damage or failures occur in an aircraft an adaptive controller takes over to help stabilize and control the aircraft. The pilots then invoke the ELP using the Flight Management Computer. The ELP provides the pilots with a ranked set of possible emergency landing sites.

Fundamentally, the ELP solves a 3D path planning problem with dynamics. It does this by constructing a probabilistic roadmap of points and edges that includes the current position of the aircraft and an approach point to every possible runway within a viable range. (This may cover hundreds of airports for an aircraft at high altitude.) A sophisticated model of risk is used to assess the probability of success for each edge in the roadmap. This model of risk takes into account:

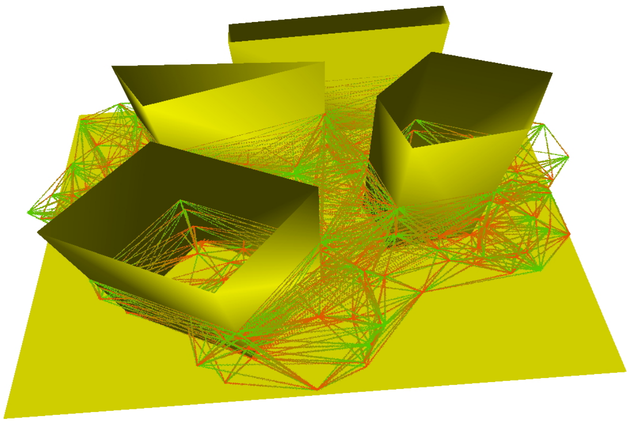


Figure 2: An example roadmap for an ELP scenario. The vertical polygons are areas of thunderstorm or other weather activity. Terrain obstacles (lower) are not shown.

- Control capabilities of the (damaged) aircraft
- Weather conditions in the area (e.g. thunderstorms, turbulence, icing)
- Ceiling, visibility and winds at each possible landing site
- Instrument approaches available at the site (if any)
- Characteristics of the landing site (runway length, width, condition)
- Emergency facilities at the site (fire, medical)
- Danger to population along the approach path

The flight envelope plays a key role in the assessment of risk for the different options. For example, if a damaged aircraft must maintain a higher airspeed than normal, additional runway length is needed, and finding a runway with a strong headwind is important in order to lower ground speed at touchdown. Similarly, if the aircraft has limited ability to bank to the right, a right crosswind or gusty conditions will be problematic, as will paths that require sharp turns to the right.

[*] Stinger Ghaffarian Technologies
[†] Mission Critical Technologies

Figure 3: The Advanced Concepts Flight Simulator (ACFS).

A^* is used to search the roadmap to find the best landing options. The g value for a given path is the product of the probabilities of success for the legs in the path. The heuristic value h is estimated using the probability of success for a direct path to the approach point for each runway (assuming no weather) times the probability of success for the approach and landing on that runway. This heuristic value is admissible, and fairly informative.

The performance of the ELP is largely a function of the number of points and edges in the roadmap. Currently, we generate 2000 points and connect them to their 200 nearest neighbors, which results in a roadmap with 400,000 edges. The A^* search typically expands about 20 percent of those edges for the scenarios we considered. With this sized roadmap, the ELP produces an ordered list of options for the pilot in under 10 seconds. This list can therefore be refreshed and updated as often as desired, to account for the aircraft movement, weather updates, or additional failures.

Our experience has been that paths generated from probabilistic roadmaps of this density can be far from optimal, and just don't look very good when displayed. This problem can be addressed by dramatically increasing the density of points and edges, but this approach also significantly increases search time. The more practical solution is to use local search to shorten and smooth paths. We do this local search by constructing a second roadmap consisting only of points along the path just found, creating a dense network of edges among those points, and re-running A^* on this reduced graph. The resulting paths are shorter, smoother, and seem more natural when displayed.

More detail about the ELP, the path planner, the risk model, and the local search can be found in (Meuleau et al. 2009; 2011a; 2011b)

Figure 4: The cockpit of the ACFS.

2. Integration

Figures 3 and 4 show the Advanced Concepts Flight Simulator (ACFS) at NASA Ames Research Center. The simulator is representative of modern glass cockpit twin engine commercial transport aircraft such as the Boeing 757, 767, and Airbus A320. The ELP was integrated into the avionics of this simulator in order to conduct experiments with teams of professional pilots with different damage scenarios and weather conditions. Unfortunately, the ACFS is not very portable, so for demonstration purposes, we use a reasonably high fidelity simulator that runs on a laptop. It includes the Primary Flight Display, Navigation Display, and Flight Management System common to modern glass cockpit aircraft. Just as in the ACFS, this simulator incorporates an adaptive controller, and has several damage models available.

A typical scenario involves starting the aircraft in cruise flight following a flight plan like that shown on the Navigation Display in Figure 5. A failure is then introduced as illustrated in the surface position display shown in Figure 6. In the case illustrated in Figure 6, the left wing is damaged and the left aileron has failed (red).

When the damage occurs, the adaptive controller takes over and stabilizes the aircraft. In the example illustrated, the adaptive controller is adding right up aileron (blue) and right spoilers (blue) to keep the aircraft from rolling left. To help the pilots understand the control limitations of the damaged aircraft, color bands are shown on the primary flight display as illustrated in Figure 7. These color bands indicate safe ranges for airspeed, bank angle and vertical speed. In this case, the aircraft must maintain a much higher speed than normal to keep sufficient airflow over the remaining aileron. The ability to bank right is also very limited.

Pilots access the ELP from the Departure/Arrival page of the Flight Management Computer as shown in Figure 8. After a brief splash screen, a set of "Emergency Pages" is displayed, showing the options ordered from lowest to highest risk. Figure 9 shows the first of four emergency pages for this scenario. Each entry shows an airport, runway, run-

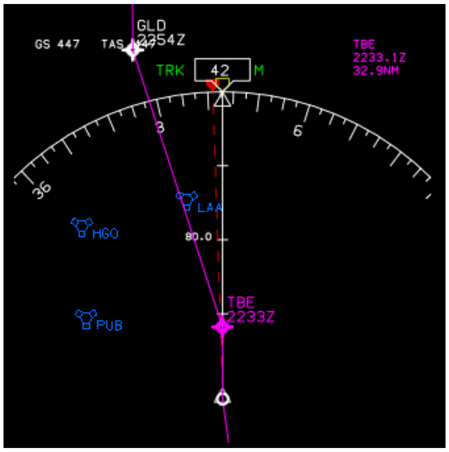


Figure 5: The Navigation Display showing the current route.

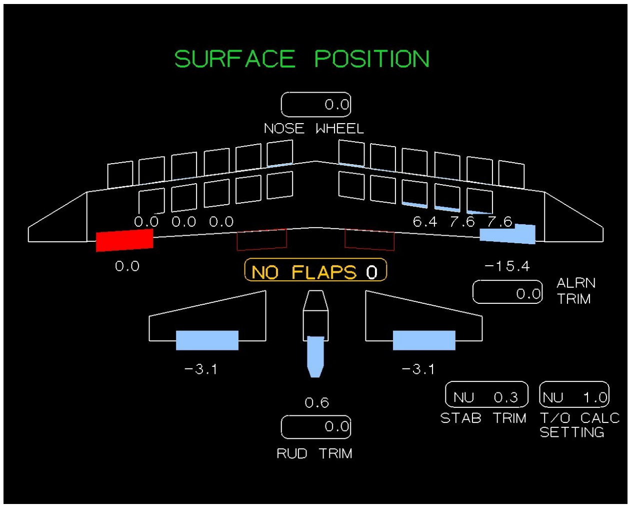


Figure 6: Surface position display showing status and deflection of control surfaces.

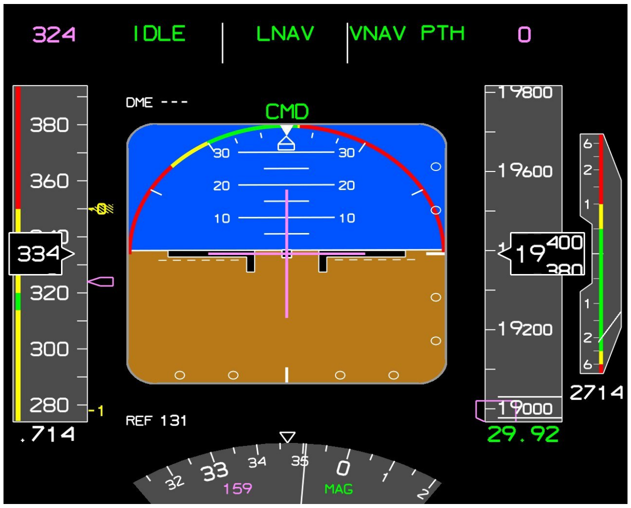
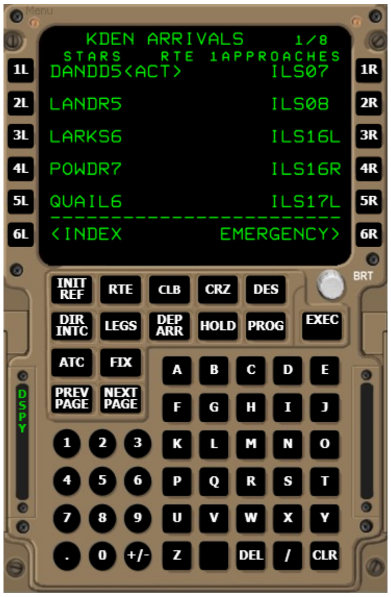
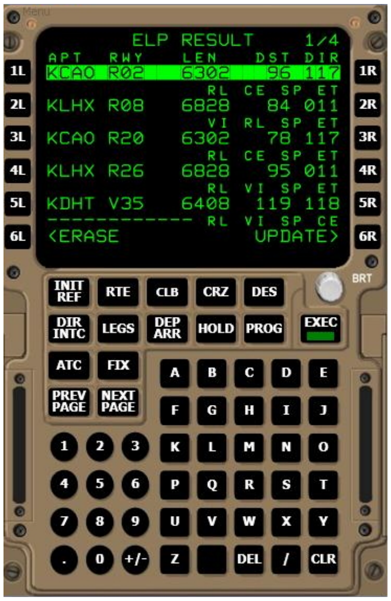


Figure 7: The Primary Flight Display (PFD) showing bank angle, pitch, airspeed, vertical speed, altitude and heading.



```
   KDEN ARRIVALS          1/8
 STARS     RTE 1APPROACHES
DANDD5<ACT>           ILS07

LANDR5                ILS08

LARKS6                ILS16L

POWDR7                ILS16R

QUAIL6                ILS17L
------------------------------
<INDEX            EMERGENCY>
```

Figure 8: The display for the Flight Management Computer showing the Departures/Arrivals page for Denver (KDEN). The emergency prompt appears next to button 6R at the lower right.



```
        ELP RESULT        1/4
APT   RWY    LEN     DST DIR
KCAO  R02    6302     96 117
              RL  CE  SP  ET
KLHX  R08    6828     84 011
              VI  RL  SP  ET
KCAO  R20    6302     78 117
              RL  CE  SP  ET
KLHX  R26    6828     95 011
              RL  VI  SP  ET
KDHT  V35    6408    119 118
------------  RL  VI  SP  CE
<ERASE               UPDATE>
```

Figure 9: The first of four emergency pages for a scenario.

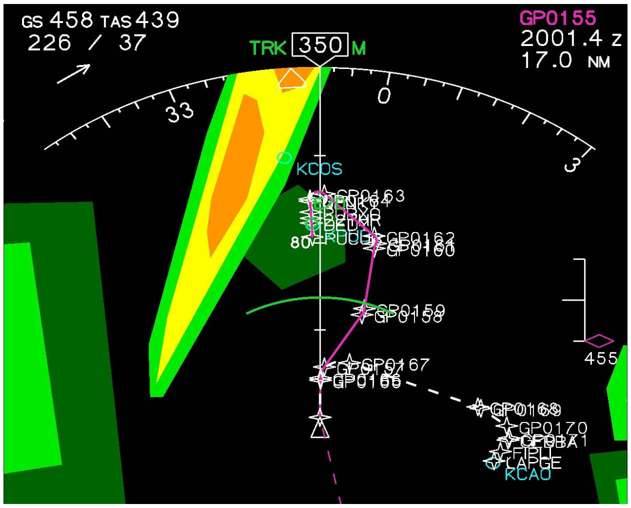


Figure 10: The Navigation Display showing both the current route (magenta) and the new route being considered (dashed white). Green, yellow, and orange areas indicate rain and thunderstorm activity.

way length, distance, and direction (magnetic bearing). The smaller symbols below each entry indicate the principle risks associated with that option; for example, RL indicates runway length is an issue, and CE indicates that the cloud ceiling is close to the minimums for the best approach to that runway. To select an entry, the button to the left of the entry is pressed. In this case, the first entry has been selected by pressing button 1L, which causes the route for that option to show up as a dashed white line on the Navigation Display, as shown in Figure 10. Pressing the EXEC key would cause the route to become the current route (solid magenta). The pilots can page through the options using the NEXT PAGE and PREV PAGE buttons as desired. To see more information about a particular option, the pilots can press the button to the right of the option, which brings up an airport information page showing runway information and the current weather at the airport (Figure 11).

Acknowledgments

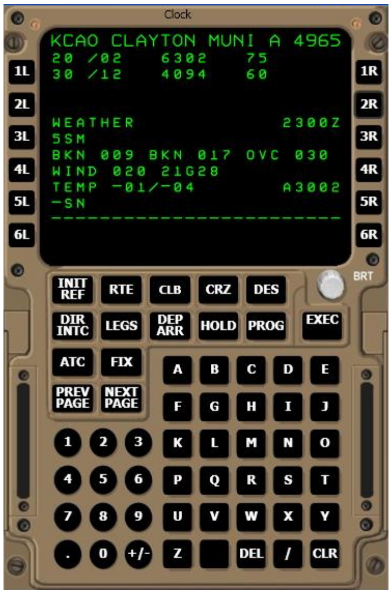
This work was supported by the NASA Aviation Safety Program and the American Reinvestment and Recovery Act. We thank John Kaneshige, Stefan Campbell, Shivanjli Sharma, Captain Mietek Steglinski, Matt Gregory and the staff of the NASA Ames Crew Vehicle Systems Research Facility for their work in helping to prepare for and conduct the simulator experiment.

References

Meuleau, N.; Plaunt, C.; Smith, D.; and Smith, T. 2009. An emergency landing planner for damaged aircraft. In *Proceedings of the Twenty First Innovative Applications of Artificial Intelligence Conference*. AAAI Press.

Meuleau, N.; Neukom, C.; Plaunt, C.; Smith, D.; and Smith, T. 2011a. The Emergency Landing Planner experiment. In *ICAPS-11 Scheduling and Planning Applications Workshop (SPARK)*.

Meuleau, N.; Neukom, C.; Plaunt, C.; Smith, D.; and Smith, T. 2011b. The Emergency Landing Planner experiment. Technical report, NASA Ames Research Center.



```
KCAO CLAYTON MUNI A 4965
20 /02    6302   75
30 /12    4094   60

WEATHER               2300Z
5SM
BKN 009 BKN 017 OVC 030
WIND 020 21G28
TEMP -01/-04          A3002
-SN
------------------------
```

Figure 11: An Airport Information page showing runways and current weather for KCAO.

Scheduling and Planning Interface for Exploration (SPIFe)

Arash Aghevli and **Alfredo Bencomo**
Autonomous Systems and Robotics Intelligent Systems
SGT, NASA Ames Research Center
Moffett Field, CA 94035

Michael McCurdy
Human-Computer Interaction Group
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

Many planning tools developed as user-facing interfaces to automated planning systems do not allow users enough flexibility to explore plans in a number of different ways, quickly understand complex sets of constraints and their implications, or experiment with different solutions without fear of losing work. Typically, such tools are architected in such a way that the user interface is integral to the underlying planning, scheduling, and simulation engine(s). The Scheduling and Planning Interface for Exploration (SPIFe) is an integrated planning and scheduling toolkit based on hundreds of hours of expert observation, use, and refinement of state-of-the-art planning and scheduling technology for several applications within NASA. It was designed from the ground up with the needs of the operational user in mind, and it presents unique solutions to a number of problems common in other commercial and homegrown systems. SPIFe has been used on the Mars Exploration Rover mission and the Phoenix Mars Lander mission, and is now being baselined for use on the next Mars Science Laboratory mission (fall of 2011). It has also been adapted as preflight planning and a real-time analysis console tool that supports all phases of planning on the International Space Station (ISS), as well as several other flight projects and analogs.

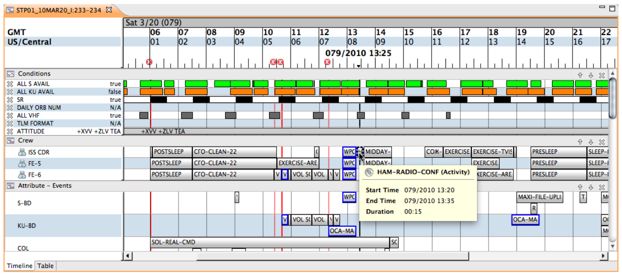
User Interface Principles and Components

The SPIFe user interface is designed to be a highly adaptable and user-customizable framework for viewing and manipulating plan and schedule data. In order to achieve this, SPIFe employs a composable, plug-in architecture based on the open source Eclipse Rich Client Platform (RCP). Eclipse provides a robust plug-in framework, and the RCP provides many fundamental user interface components, such a tabbed ”workbench” that allows users to manipulate views and editors to display the information most relevant to the task at hand. The following sections describe a number of SPIFe views and editors that can be combined (or omitted) depending on the needs of a particular planning application.

The Timeline

One of the central components of the SPIFe framework, the timeline (Figure 1) provides a traditional time-based representation of a plan. Activities appear as bars that vary in width according to when they're scheduled. Timeline rows are highly configurable: which rows are displayed, their ordering, bar figure look and feel, and row criteria (which determine whether a given activity appears on a given row) can all be modified in a descriptor file for a given application.

The goal of this extensive configurability is to provide an appropriately detailed representation of a complex schedule that is responsive to the needs of a particular user – suppressing details that are not likely to be relevant or understood and presenting others at a level of abstraction appropriate for a user to make planning decisions on his or her own, or easily understand the results of an automated planner. For example, a typical Mars rover plan may include hundreds of individual activities, each with copious metadata – parameters, notes, and results of resource estimates. A user may choose to use a hierarchical grouping to associate activities of similar scientific intent or using similar spacecraft hardware, then work with these higher level activity groups on the timeline when making planning decisions or presenting the schedule to other stakeholders.



HAM-RADIO-CONF (Activity)

Start Time 079/2010 13:20
End Time 079/2010 13:35
Duration 00:15

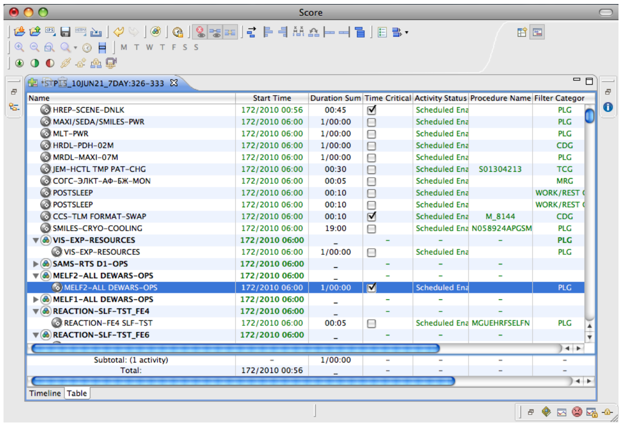
Figure 1: The SPIFe Timeline

All activities can be edited directly via drag-and-drop, and the timeline also provides several feature such as multiple selection, feedback during editing operations, and full support for multiple levels of Undo and Redo to allow users to freely explore multiple solutions. The goal of the SPIFe timeline is to capitalize on user familiarity with common visual editing paradigms where possible (e.g. manipulating figures in drawing tools like Visio or Powerpoint) in order to remain approachable by non-experts.

Internal SPIFe constraint checkers as well as external systems can provide detailed temporal violations and have them displayed within the context of the timeline. Additional information such as violation culprits can be identified visually via activity borders. Tooltips can be enabled to show a configurable level of greater detail on violations or the activities themselves, as well as providing quick access to common fixes for temporal constraint violations or other schedule defects.

The Table Editor

In addition to the timeline editor, SPIFe provides a tabular representation of the activities and groups in the plan. The Table Editor (Figure 2) is useful for displaying a large number of activities, and is especially useful for plans that are sparsely populated (few events over long periods of time) where a timeline display would be mostly empty for a given time range. The Table Editor can be configured with columns representing each piece of activity metadata, including basic start time and duration information as well as details of resource requirements or per-activity resource consumption predicts.



*STP15_10JUN21_7DAY:326-333

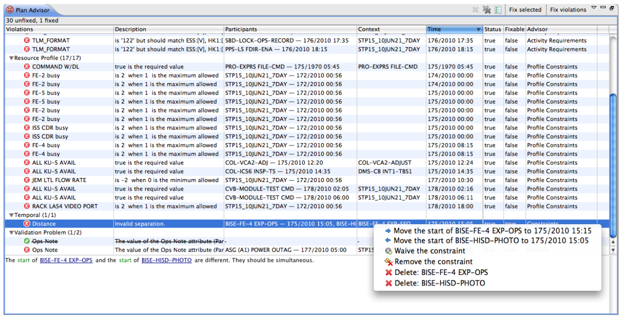
Name	Start Time	Duration Sum	Time Critical	Activity Status	Procedure Name	Filter Categor
HREP-SCENE-DNLK	172/2010 00:56	00:45	☑	Scheduled Ena		PLG
MAXI/SEDA/SMILES-PWR	172/2010 06:00	1/00:00	□	Scheduled Ena		PLG
MLT-PWR	172/2010 06:00	1/00:00	□	Scheduled Ena		PLG
HRDL-PDH-02M	172/2010 06:00	1/00:00	□	Scheduled Ena		CDG
MRDL-MAXI-07M	172/2010 06:00	1/00:00	□	Scheduled Ena		PLG
JEM-HCTL TMP PAT-CHG	172/2010 06:00	00:30	□	Scheduled Ena	S01304213	TCG
СОГС-ЭЛКТ-АФ-БЖ-MON	172/2010 06:00	00:05	□	Scheduled Ena		MRG
POSTSLEEP	172/2010 06:00	00:10	□	Scheduled Ena		WORK/REST (
POSTSLEEP	172/2010 06:00	00:10	□	Scheduled Ena		WORK/REST (
CCS-TLM FORMAT-SWAP	172/2010 06:00	00:10	☑	Scheduled Ena	M_8144	CDG
SMILES-CRYO-COOLING	172/2010 06:00	19:00	□	Scheduled Ena	N058924APGSM	PLG
▼ **VIS-EXP-RESOURCES**	**172/2010 06:00**	_	-	-	-	**PLG**
VIS-EXP-RESOURCES	172/2010 06:00	1/00:00	□	Scheduled Ena		PLG
▶ **SAMS-RTS D1-OPS**	**172/2010 06:00**	_	-	-	-	
▼ **MELF2-ALL DEWARS-OPS**	**172/2010 06:00**	_	-	-	-	
MELF2-ALL DEWARS-OPS	172/2010 06:00	1/00:00	☑	Scheduled Ena		PLG
▶ **MELF1-ALL DEWARS-OPS**	**172/2010 06:00**	_	-	-	-	
▼ **REACTION-SLF-TST_FE4**	**172/2010 06:00**	_	-	-	-	
REACTION-FE4 SLF-TST	172/2010 06:00	00:05	□	Scheduled Ena	MGUEHRFSELFN	PLG
▼ **REACTION-SLF-TST_FE6**	**172/2010 06:00**	_	-	-	-	
Subtotal: (1 activity)	-	1/00:00	-	-	-	-
Total:	172/2010 00:56	_	-	-	-	-

Figure 2: The SPIFe Table Editor

The majority of observed planning processes involve some form of plan integration. For example, the science team on a Mars mission may be broken up into theme groups around instruments or areas of scientific intent. Each team may build partial plans in parallel, and then feed them back in to an integrated plan for the spacecraft. In order to facilitate this merging process, users can open as many plan fragments as needed and simply drag and drop or copy/paste from one editor to another. For more sophisticated merge operations that happen on a routine basis (such as integrating international partner inputs into a plan for the International Space Station), automated merge and integrate capability can be developed.

The Plan Advisor

One of the fundamental design principles of the SPIFe toolkit is that the user's hand should not be forced by any integrated automated planning system. As a result, much of the feedback from the native constraint and resource engine as well as feedback from external systems is presented in a view called the Plan Advisor (Figure 3). The concept behind the Plan Advisor is that the human is in control of the plan, but he or she may selectively invoke help from automated systems.



30 unfixed, 1 fixed

Violations	Description	Participants	Context	Time	Status	Fixable	Advisor
TLM_FORMAT	is '122' but should match ESS:[V], HK1:[	SBD-LOCK-OPS-RECORD — 176/2010 17:35	STP15_10JUN21_7DAY	176/2010 17:35	true	false	Activity Requirements
TLM_FORMAT	is '122' but should match ESS:[V], HK1:[	PPS-LS FDIR-ENA — 176/2010 18:15	STP15_10JUN21_7DAY	176/2010 18:15	true	false	Activity Requirements
▼Resource Profile (17/17)							
COMMAND W/DL	true is the required value	PRO-EXPRS FILE-CMD — 175/1970 05:45	PRO-EXPRS FILE-CMD	175/1970 05:45	true	false	Profile Constraints
FE-2 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		174/2010 00:00	true	false	Profile Constraints
FE-2 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		174/2010 00:00	true	false	Profile Constraints
FE-5 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
FE-5 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
FE-2 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
FE-2 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
ISS CDR busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
ISS CDR busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 00:00	true	false	Profile Constraints
FE-4 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 08:15	true	false	Profile Constraints
FE-4 busy	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		175/2010 08:15	true	false	Profile Constraints
ALL KU-S AVAIL	true is the required value	COL-VCA2-ADJ — 175/2010 12:20	COL-VCA2-ADJUST	175/2010 12:24	true	false	Profile Constraints
ALL KU-S AVAIL	true is the required value	COL-ICS6 INSP-TS — 175/2010 14:35	DMS-CB INT1-TBS1	175/2010 14:35	true	false	Profile Constraints
JEM LTL FLOW RATE	is -2 when 0 is the minimum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		177/2010 10:30	true	false	Profile Constraints
ALL KU-S AVAIL	true is the required value	CVB-MODULE-TEST CMD — 178/2010 02:05	STP15_10JUN21_7DAY	178/2010 02:16	true	false	Profile Constraints
ALL KU-S AVAIL	true is the required value	CVB-MODULE-TEST CMD — 178/2010 06:00	STP15_10JUN21_7DAY	178/2010 06:11	true	false	Profile Constraints
RACK LAS4 VIDEO PORT	is 2 when 1 is the maximum allowed	STP15_10JUN21_7DAY — 172/2010 00:56		178/2010 18:00	true	false	Profile Constraints
▼Temporal (1/1)							
Distance	Invalid separation.	BISE-FE-4 EXP-OPS — 175/2010 15:05, BISE-HI	BISE-FE-4 EXP-SEQ	175/2010 15:05	true	true	Constraints
▼Validation Problem (1/2)							
~~Ops Note~~	~~The value of the Ops Note attribute (Par~~	-	-				
Ops Note	The value of the Ops Note attribute (Par	ASG (A1) POWER OUTAG — 177/2010 05:00	STP15				

The start of BISE-FE-4 EXP-OPS and the start of BISE-HISD-PHOTO are different. They should be simultaneous.

- Move the start of BISE-FE-4 EXP-OPS to 175/2010 15:15
- Move the start of BISE-HISD-PHOTO to 175/2010 15:05
- Waive the constraint
- Remove the constraint
- Delete: BISE-FE-4 EXP-OPS
- Delete: BISE-HISD-PHOTO

Figure 3: The SPIFe Plan Advisor

In most cases feedback is presented to the user in realtime after each plan edit. If a violation is determined to be fixable, either by native code or an external engine, users are presented with a context menu containing common fixes. If more extensive reasoning or search is required, users can invoke the capabilities of external systems via "Fix Violations" commands which are also invoked from the Advisor.

In many cases violations are deemed acceptable, either due to a one-time exception, or more commonly an error or omission with the model or constraints themselves. In these cases, users can waive the violation and provide rationale. These waivers and rationale are persisted with plan data so an audit history is always preserved.

Resource Modeling

SPIFe has the capability to display resource usage effects that are derived from the schedule and visualize resource modeling information of varying kinds from coarse approximations to extremely high resolution simulation data. It also supports a multitude of higher fidelity simulation engines to display things like power, geometry (e.g. position of sun relative to spacecraft), or data usage. The results of external modeling tools are transferred seamlessly to the SPIFe toolkit for display in the context of the planning session: alongside the timeline, in columns in the table editor, in fields in an inspector pane associated with each activity, and in the Plan Advisor if necessary. This allow users to immediately see the effect of plan changes within the same context and debug issues that potentially result from them.

External Toolkit Integration

The implementation of SPIFe was designed from the ground up with integration of external planning and scheduling systems in mind. While some capabilities exist natively within SPIFe, most domains that have been encountered utilize high fidelity simulation and automated planning engines. Robotic Mars and International Space Station missions alike utilize agency and/or center wide standard modeling engines that simulate geometric, electrical and thermal fluctuations of the environment and physical hardware for presentation

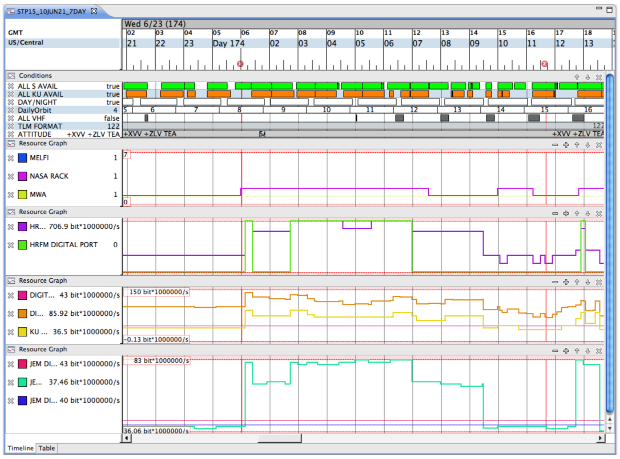


Figure 4: The SPIFe timeline displaying several predict plots from a data model

to the planner. Planning engines have also been integrated that allow external systems to propose modifications to the currently editable plan. Such systems reconcile temporal and resource constraints and can be as complex as requiring distributed system architecture, or as simple as scripts that reduce planning redundancies within the product.

The integration with such systems has been made possible through client side translation of internal SPIFe models to external system models. These models are then communicated through RESTful interfaces, JNI, XML-RPC and/or the spawning of new processes to evaluate the plan and return simulated resource values and/or proposals for plan modifications which can then be applied by the user while maintaining the undo/redo functionality that allows users to back out changes if necessary to correct overlooked constraints and or resource allocations.

Each deployment of SPIFe has always come with unique challenges and thus unique engines and systems to integrate with. For the Phoenix Mars Lander, SPIFe integrated with the Europa planning engine to fix temporal violations, APcore which provided high-fidelity modeling of data acquisition and transfer, and the JPL-developed Multi-Mission Power Analysis Tool (MMPAT) for high-fidelity power and thermal modeling. For the International Space Station Power simulation product, SPIFe interfaces with numerous tools, bringing together numerous different modeling and simulation engines in a single, consistent user interface. These include the Spacecraft Electrical Equipment Database (SEED), Electrical Power Load Model (EPLM), the Battery and Solar Array Model (BSAM), Flight Dynamic Planning and Analysis (FDPA), Robotic Shadowing Calculator (RSC) and Solar Array Constraint Engine (SACE).

Integration with all of these high fidelity engines not only allow for the continued use of domain specific systems, but allow SPIFe to leverage years of usability testing to make the presentation of such capabilities intuitive, while not compromising on the computational requirements to run a safe and efficient mission.

Architectural Foundations

The modeling capability in SPIFe employs a widely utilized modeling framework called the Eclipse Modeling Framework (EMF). It is primarily an implementation of the Object Management Group's (OMG) Meta Object Facility (MOF). This capability allows SPIFe to tap into an extensive library of support services that support the generation of metamodels to describe domains using UML based tools, XML schemas, database tables, by hand using Eclipse based tooling, as well as a host of other techniques that continue to evolve.

This standardization increases flexibility while reducing the overhead that comes with use of planning and scheduling tools. Many deployments start with a careful analysis of the high level information planners wish to specify, and codify them into models. Many times, these specifications already exist in the form of database or XMl schemas, in which case the use of the EMF/ MOF capabilities make integration with existing systems trivial. If no such standards currently exist, industry standard tools can be utilized to create such models quickly.

Behavior of SPIFe planning models is specified through the use of various modeling languages as defied by the automated systems that SPIFe utilizes. In many instances, the native SPIFe capabilities are used which leverages JavaScript to take the metamodels and specify the effects, constraints and conflict resolution strategies once added to the plan. The use of JavaScript itself allows for the leveraging of a great deal of shared development resources in terms of tooling and documentation support.

In most cases however, the specific deployments of SPIFe in domains use external engines that typically have their own specific domain specific languages (DSL) to define the metamodels. In such cases, the EMF / MOF capabilities are only utilized to allow the UI to be configured for data entry and visualization of the information. The data is thus sent and returned to and from planning and scheduling engines asynchronously, keeping both the automated and manual panner in the loop at all times.

Concluding Remarks

Missions understandably set relatively high bars when it comes to stability, control, efficiency, and transparency in their operations processes. This may be especially true in missions with tight tactical planning cycles. Here, plans must be assembled quickly, and it must be widely understood why a plan has been assembled the way it has before a commitment is made to sequence it and execute it. The addition of automated planning technology then further accelerates the planning process. The focus on mixed initiative planning, where plan flaws are noted and repair assistance is provided, greatly contributes to transparency and control, without which rapid planning in a tactical operations context is far less useful. The ability to work with plans that are invalid from the perspective of the planning model allows users to incrementally build and repair a plan they understand and can explain.

In addition, experience suggests there will always be nuances, exceptions or changes to how a mission chooses to operate a spacecraft, and there isn't time during the tactical cycle to bring the existing planner model into agreement with the ground truth about the rover as understood by the operators. Having close control over the modifications the planning technology suggests for the plan is crucial in these situations.

Acknowledgments

None of the work presented in this demo would have become a reality without the dedication and passion of several remarkable teams. We would like to thank the JPL, JSC, and Ames development and design teams for their numerous contributions to the SPIFe toolkit.

References

Bresina J., Jonsson A., Morris P., and Rajan K. 2005a. Activity Planning for the Mars Exploration Rovers, Fourteenth International Conference on Automated Planning and Scheduling, Monterey, 2005, pp. 40-49.

Bresina J., Jonsson A., Morris P., Rajan K. 2005b. Mixed- Initiative Planning in MAPGEN: Capabilities and Shortcomings. Workshop on Mixed-Initiative Planning and Scheduling, ICAPS05, 2005.

Bresina J., Morris P. 2006. Mission Operations Planning: Beyond MAPGEN, Second IEEE International Conference On Space Mission Challenges for Information Technology (SMCIT), Pasadena, 2006.

Fox J. M., Norris J. S., Powell M. W., Rabe K. J., Shams, K. S. 2006. "Advances in Distributed Operations and Mission Activity Planning for Mars Surface Exploration", AIAA SpaceOps 2006, Rome, June 19-23, 2006.

Frank J., Jnsson A., 2003. Constraint-Based Interval and Attribute Planning. Journal of Constraints Special Issue on Constraints and Planning.

Maldague P., Ko A., Page D., and Starbird T., 1998. APGEN: A multi-mission semi-automated planning tool. First International NASA Workshop on Planning and Scheduling, Oxnard, CA, 1998.

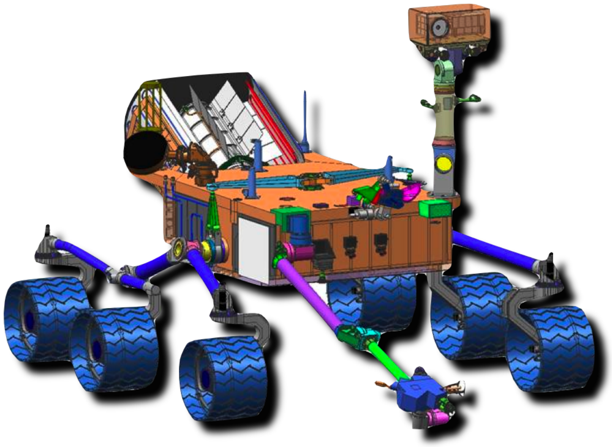
McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., and Vera, A. 2006. Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed- fidelity success. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (Montral, Qubec, Canada, April 22 - 27, 2006).

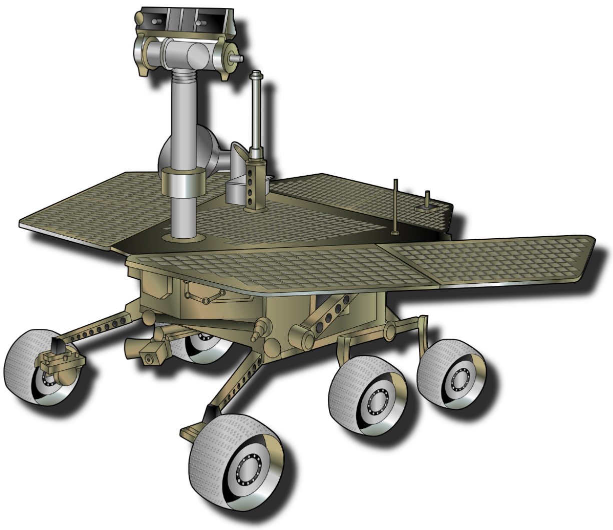
Norris J. S., Powell M. W., Fox J. M., Rabe K. J., Shu I. 2005a. "Science Operations Interfaces for Mars Surface Exploration," 2005 IEEE Conference on Systems, Man, and Cybernetics, Big Island, HI., October 15, 2005.

Norris J. S., Powell M. W., Vona M. A., Backes, P. G., Wick, J. V. 2005b. "Mars Exploration Rover Operations with the Science Activity Planner," IEEE International Conference on Robotics and Automation, April 2005.

Tollinger, I., McCurdy, M., Vera, A. H., and Tollinger, P. 2004. Collaborative knowledge management supporting mars mission scientists. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (Chicago, Illinois, USA, November 06 - 10, 2004).

Aghevli A, Bachmann A., Bresina J., Greene K., Kanefsky B., Kurien J., McCurdy M., Morris P., Pyrzak G., Ratterman C., Vera A., Wragg S., 2006. "Planning Applications for Three Mars Missions with Ensemble", AIAA SpaceOps 2006, California, June 2006.





A New Approach to Conformant Planning via Classical Planners

Khoi Nguyen and **Vien Tran** and **Tran Cao Son** and **Enrico Pontelli**
Computer Science Department
New Mexico State University
Email: {knguyen,vtran,tson,epontell}@cs.nmsu.edu

Abstract

In this paper, we introduce a new approach to conformant planning via classical planners. We view a conformant planning problem as a set of classical planning problems, called *sub-problems*, and solve it using a generate-and-complete algorithm. Key to this algorithm is a procedure which takes a solution of a sub-problem and generates a solution for other sub-problems. We implement this algorithm in a new planner, called CPCL and evaluate it empirically against state-of-the-art conformant planners using various benchmarks. The experimental results show that CPCL is superior to other planners in most benchmarks, both in performance and in scalability.

Introduction

Conformant planning is the problem of computing a sequence of actions that achieves a goal in presence of incomplete information about the initial state (Smith and Weld 1998). By definition, conformant planning searches for the plan in the belief state space. Due to the incomplete information, the belief state usually has large size which leads to difficulty in searching for the solution. Thus one way to address the problem is to translate the conformant planning problem to a classical planning problem which has been done by `t0` (Palacios and Geffner 2006).

The idea of using a classical planning system to solve a non-classical planning problem has been applied to other types of planning problems such as probabilistic planning. FF-Replan (Yoon *et al.* 2007), the winner of the 2004 IPC, which solves a probabilistic planning problem by (*i*) translating the problem into a classical planning problem, (*ii*) computing a solution using a classical planner (FF), and (*iii*) replanning whenever necessary.

It is interesting to contrast the approaches adopted in `t0` and FF-Replan. While the translation employed by `t0` could produce a new problem whose size is exponential in the size of the original one (if completeness is required), and thus making the problem more difficult, the *determinizing* process of FF-Replan simplifies the original problem by removing all information related to non-determinicity. This raises the interesting question of whether an alternative approach to `t0`, perhaps in a similar spirit to that of FF-Replan, could produce similar results in conformant planning. It is clear that the algorithm of FF-Replan cannot be applied to conformant planning, since conformant planning does not interleave planning and execution.

In this paper, we develop a new approach to conformant planning using classical planners. We implement the idea in a system, called CPCL, and evaluate it against state-of-the-art conformant planners using several benchmarks. The experimental results show that the new planner performs exceptionally well in almost all domains and scales up better than other planners.

Conformant Planning Problem

A conformant planning problem P is specified by a tuple $\langle F, O, I, G \rangle$, where F is a set of propositions, O a set of action descriptions, I a set of formulae describing the initial state of the world, and G a formula describing the goal.

A *literal* is a proposition $p \in F$ or its negation $\neg p$. $\overline{\ell}$ denotes the complement of the literal ℓ, and it is defined as $\overline{\ell} = \neg \ell$, where $\neg\neg p = p$ for $p \in F$. For a set of literals L, $\overline{L} = \{\overline{\ell} \mid \ell \in L\}$; and L is often used to represent $\wedge_{\ell \in L} \ell$.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A *state* s is a consistent and *complete* set of literals, i.e., s is consistent, and for each $p \in F$, either $p \in s$ or $\neg p \in s$. A *belief state* is a set of states. A set of literals X satisfies a literal ℓ (resp. a set of literals Y) iff $\ell \in X$ (resp. $Y \subseteq X$).

Each action a in O is associated with a precondition, denoted by $pre(a)$, and a set of conditional effects of the form $\psi \rightarrow \ell$ (denoted by $a : \psi \rightarrow \ell$), where $pre(a)$ and ψ are sets of literals and ℓ is a literal. We often write $a : \psi \rightarrow \ell_1, \ldots, \ell_k$ as a shorthand for the set $\{a : \psi \rightarrow \ell_1, \ldots, a : \psi \rightarrow \ell_k\}$.

The initial state I is a collection of literals, `one-of` clauses (each of the form `one-of`$(\psi_1, \ldots, \psi_n)$), and `or` clauses (each of the form `or`$(\psi_1, \ldots, \psi_m)$) where each ψ_i is a set of literals.

A set of literals X satisfies the `one-of` clause `one-of`$(\psi_1, \ldots, \psi_n)$ if there exists some i, $1 \leq i \leq n$, such that $\psi_i \subseteq X$ and for every $j \neq i$, $1 \leq j \leq n$, $\overline{\psi_j} \cap X \neq \emptyset$. X satisfies the `or` clause `or`$(\psi_1, \ldots, \psi_m)$ if there exists some $1 \leq i \leq m$ such that $\psi_i \subseteq X$.

By $ext(I)$ we denote the set of all states satisfying every literal in I, every `one-of` clause in I, and every `or` clause in I (e.g., if F=$\{g, f, h\}$ and I=$\{$`or`(g, h), `one-of`$(f, h)\}$ then $ext(I) = \{\{g, h, \neg f\}, \{g, \neg h, f\}, \{\neg g, h, \neg f\}\}$).

The goal G is a collection of literals and `or` clauses.

Given a state s and an action a, a is executable in s if $pre(a) \subseteq s$. A conditional effect $a : \psi \rightarrow l$ is applicable in s if $\psi \subseteq s$. The set of effects of a in s, denoted by $e_a(s)$, is defined as: $e_a(s) = \{l \mid a : \psi \rightarrow l \in O$ is applicable in $s\}$. The execution of a in a state s results in a successor state $succ(a, s)$, where $succ(a, s) = (s \cup e_a(s)) \setminus \overline{e_a(s)}$ if a is executable in s, and $succ(a, s) =$ **failed**, otherwise. Using this function, we define $\widehat{succ}$ for computing the state resulting from the execution of a sequence of actions $\alpha = [a_1, \ldots, a_n]$: $\widehat{succ}(\alpha, s) = s$ if $n = 0$; and $\widehat{succ}(\alpha, s) = succ(a_n, \widehat{succ}(\beta, s))$ if $n > 0$ where $\beta = [a_1, \ldots, a_{n-1}]$ and $\widehat{succ}(\gamma, \textbf{failed}) \overset{def}{=}$ **failed** for any sequence of actions γ. For a belief state S and action sequence α, let $\widehat{succ}^*(\alpha, S) = \{\widehat{succ}(\alpha, s) \mid s \in S\}$ if $\widehat{succ}(\alpha, s) \neq$ **failed** for every $s \in S$; and $\widehat{succ}^*(\alpha, S) =$ **failed**, otherwise. α is a *solution* of P iff $\widehat{succ}^*(\alpha, ext(I)) \neq$ **failed** and G is satisfied in every state belonging to $\widehat{succ}^*(\alpha, ext(I))$.

Conformant Planning using a Classical Planner—An Intuition

In this section, we present our idea of how to use a classical planner to solve conformant planning problems. Let us illustrate our idea in an example.

Example 1. Let us consider a small instance (denoted by P_1) of the coin problem from the IPC 2008 (Bryce and Buffet 2008). In this problem, we have one elevator e_0 which can move between floors f_0 and f_1 if one of the actions *go_up* or *go_down* is performed, depending on the location of the elevator. Each floor has two positions p_0 and p_1. An agent can enter (or exit) the elevator by using the action *step_in* (or *step_out*). The agent can also move between positions on the same floor by using the actions *move_left* and *move_right*. If the agent is at the same position as a coin, he can collect the coin by using the action *collect*.

There is one coin, denoted by c_0, whose initial location is only partially known: the coin is on the floor f_1 but it is not known whether it is at the position p_0 or p_1. Furthermore, the elevator's location is initially unknown: it can be at f_0 or f_1. Initially, the agent is at the position p_0 of floor f_0.

The goal is of the problem is to collect the coin c_0—denoted by the fluent $have(c_0)$.

Let us explore the encoding $P_1 = \langle F, O, I, G \rangle$. In this domain, the set of propositions F contains the following propositions:[1]

- $at(f, p)$: the agent is at position p of floor f,
- $in(e, f)$: the elevator is at the floor f,
- $coin\_at(c, f, p)$: the coin c is at position p of the floor f,
- $inside(e)$: the agent is inside the elevator e,
- $have(c)$: the agent has the coin.

where $f \in \{f_0, f_1\}$, $p \in \{p_0, p_1\}$, $e = e_0$, and $c = c_0$. The set of actions O with their conditional effects is given next:

$$\begin{array}{l}
go\_up(e, f_0, f_1) : in(e, f_0) \rightarrow in(e, f_1), \neg in(e, f_0) \\
go\_down(e, f_1, f_0) : in(e, f_1) \rightarrow in(e, f_0), \neg in(e, f_1) \\
step\_in(e, f, p) : in(e, f) \rightarrow inside(e), \neg at(f, p) \\
step\_out(e, f, p) : in(e, f) \rightarrow at(f, p), \neg inside(e) \\
move\_left(f, p_1, p_0) : true \rightarrow at(f, p_0), \neg at(f, p_1) \\
move\_right(f, p_0, p_1) : true \rightarrow at(f, p_1), \neg at(f, p_0) \\
collect(c_0, f, p) : coin\_at(c_0, f, p) \rightarrow have(c_0), \neg coin\_at(c_0, f, p)
\end{array}$$

where $f \in \{f_0, f_1\}$, and $p \in \{p_0, p_1\}$. In addition,

$$\begin{array}{ll}
pre(go\_up(e, f_0, f_1)) & = \{\} \\
pre(go\_down(e, f_1, f_0)) & = \{\} \\
pre(step\_in(e, f, p)) & = \{at(f, p)\} \\
pre(step\_out(e, f, p)) & = \{inside(e)\} \\
pre(move\_left(f, p, p')) & = \{at(f, p)\} \\
pre(move\_right(f, p, p')) & = \{at(f, p)\} \\
pre(collect(c_0, f, p)) & = \{at(f, p)\}
\end{array}$$

The initial state of the problem can be given by $I = I^d \cup I^o$ where $I^d = \{at(f_0, p_0)\}$ and

$$\begin{array}{c}
I^o = \{\texttt{one-of}(coin\_at(c_0, f_1, p_0), coin\_at(c_0, f_1, p_1)), \\
\texttt{one-of}(in(e_0, f_0), in(e_0, f_1))\}.
\end{array}$$

Finally, the goal of the problem is given by $G = \{have(c_0)\}$. Let

$$\begin{array}{l}
u_0 = \{at(f_0, p_0), coin\_at(c_0, f_1, p_0), in(e_0, f_0)\} \\
u_1 = \{at(f_0, p_0), coin\_at(c_0, f_1, p_1), in(e_0, f_0)\} \\
u_2 = \{at(f_0, p_0), coin\_at(c_0, f_1, p_0), in(e_0, f_1)\} \\
u_3 = \{at(f_0, p_0), coin\_at(c_0, f_1, p_1), in(e_0, f_1)\}
\end{array}$$

Define $s_i = comp(u_i)$. We have that $ext(I) = \{s_0, s_1, s_2, s_3\}$. One of the solutions to this problem is:

$$\alpha = \left[\begin{array}{l}
go\_down(e_0, f_1, f_0), step\_in(e_0, f_0, p_0), \\
go\_up(e_0, f_0, f_1), step\_out(e_0, f_1, p_0), \\
collect(c_0, f_1, p_0), move\_right(f_1, p_0, p_1), \\
collect(c_0, f_1, p_1)
\end{array}\right]$$

□

Let us introduce the notion of sub-problem.

Definition 1. *Let $P = \langle F, O, I, G \rangle$ be a conformant planning problem. For every $s \in ext(I)$, the planning problem $P(s) = \langle F, O, s, G \rangle$ is called a* sub-problem *of P.*

Clearly, for every $s \in ext(I)$, $P(s)$ is a classical planning problem. It is obvious that solution of P can be founded by selecting (randomly) a sub-problem $P(s)$ of P and repeatedly (**i**) computing a solution α of $P(s)$; and (**ii**) testing if α is a solution of P until a solution of P is found. Even though this process is theoretically sound, such a brute-force computation may not be practical for different reasons. First, the set of solutions of $P(s)$ is generally infinite and thus generating all solutions is impractical. Second, for efficiency and space reasons, most state-of-the-art planners use heuristics and remove some parts of the search space (non-optimal planners avoid exploring the same state twice while optimal planners ignore paths which violate some criteria, e.g., cost of current path is greater than an established threshold). Third, the process ignores the relationships among the sub-problems which are often useful in solving the problem. Inspired by FF-Replan, we develop an algorithm for conformant planning using a modification of the steps (**i**)-(**ii**).

[1] For simplicity, we omit the predicate $shaft(e, p)$, $dec\_f(f, f')$ and $dec\_p(p, p')$ that denotes the spatial relation between elevators, positions and floors as they are static and will be compiled away by the preprocessor of most planners.

Although every solution α of P is a solution of $P(s)$, it is often the case that we can find a subsequence[2] α_s of α such that α_s is a solution of $P(s)$. For example, for the problem P_1 in Example 1, the following sub-sequences α_{s_i} of α, are solutions of $P(s_i)$:

$$\begin{aligned}
\alpha_{s_0} &= [\mathit{step\_in}(e_0,f_0,p_0), \mathit{go\_up}(e_0,f_0,f_1),\\
&\qquad \mathit{step\_out}(e_0,f_1,p_0), \mathit{collect}(c_0,f_1,p_0)]\\
\alpha_{s_1} &= [\mathit{step\_in}(e_0,f_0,p_0), \mathit{go\_up}(e_0,f_0,f_1),\\
&\qquad \mathit{step\_out}(e_0,f_1,p_0), \mathit{move\_right}(f_1,p_0,p_1),\\
&\qquad \mathit{collect}(c_0,f_1,p_1)]\\
\alpha_{s_2} &= [\mathit{go\_down}(e_0,f_1,f_0), \mathit{step\_in}(e_0,f_0,p_0),\\
&\qquad \mathit{go\_up}(e_0,f_0,f_1), \mathit{step\_out}(e_0,f_1,p_0),\\
&\qquad \mathit{collect}(c_0,f_1,p_0)]\\
\alpha_{s_3} &= [\mathit{go\_down}(e_0,f_1,f_0), \mathit{step\_in}(e_0,f_0,p_0),\\
&\qquad \mathit{go\_up}(e_0,f_0,f_1), \mathit{step\_out}(e_0,f_1,p_0),\\
&\qquad \mathit{move\_right}(f_1,p_0,p_1), \mathit{collect}(c_0,f_1,p_1)]
\end{aligned}$$

Thus, one way of solving conformant problem is to modify a solution α_s of a sub-problem $P(s)$ of P—by adding actions—so that it becomes a solution of P, as shown in the next example.

Example 2. Let us consider the problem P_1 from Example 1. Let us assume that the classical planner selects s_0 from $\mathit{ext}(I)$ and generates $\alpha_{s_0} = [\mathit{step\_in}(e_0,f_0,p_0)$, $\mathit{go\_up}(e_0,f_0,f_1)$, $\mathit{step\_out}(e_0,f_1,p_0)$, $\mathit{collect}(c_0,f_1,p_0)]$ as its first solution.

α_{s_0} is not a solution of $P_1(s_1)$. However, α_{s_0} and α_{s_1} share the first three actions and α_{s_0} is executable in s_1. Furthermore, for $s'_1 = \widehat{succ}(s_1, \alpha_{s_0})$, we have that[3] $\alpha_{01}{=}\alpha_{s_0} \circ \beta$, where $\beta = [\mathit{move\_right}(f_1,p_0,p_1), \mathit{collect}(c_0,f_1,p_1)]$, is a solution of $P_1(s_1)$.

Observe that α_{01} is also a solution of $P_1(s_0)$. Thus, α_{01} is a solution of the planning problem $\langle P, O, \{\texttt{or}(s_0,s_1)\}, G\rangle$.

Let us consider $P_1(s_2)$. Checking to see whether α_{01} is a solution of $P_1(s_2)$ reveals that its first action, $\mathit{step\_in}(e_0,f_0,p_0)$, is executable in s_2; however, one of the effects of this action, $\neg at(f_0,p_0)$, is not contained in $succ(\mathit{step\_in}(e_0,f_0,p_0), s_2)$ because the precondition $in(e_0,f_0)$ of the conditional effect $\mathit{step\_in}(e_0,f_0,p_0)$: $in(e_0,f_0) \rightarrow \neg at(f_0,p_0)$ is not satisfied in s_2. In order to achieve this condition for $\mathit{step\_in}(e_0,f_0,p_0)$ from s_2, we should execute first the action $\mathit{go\_down}(e_0,f_1,f_0)$. Let $\alpha_{012} = [\mathit{go\_down}(e_0,f_1,f_0)] \circ \alpha_{01}$. We can verify that α_{012} is a solution of $P_1(s_2)$. Moreover, we can also see that α_{012} is a solution of $P_1(s_0)$, $P_1(s_1)$, and $P_1(s_3)$. In other words, α_{012} is a solution of P_1. Observe that α_{012} is identical to the solution given in Example 1. □

The above example shows that it is possible to use a classical planner and search in the original state space of a conformant planning problem for a solution by repeating the following two steps until a solution is found:

- Compute a solution α_s of a sub-problem $P(s)$ of P, and
- Incrementally repair α_s to meet the needs of other subproblems of P.

CpCl—A New Conformant Planner

We now describe a novel algorithm, called CpCl, which solves a conformant planning problem by solving several classical planning problems.

Algorithm

Alg. 1 shows the main search algorithm of the planner CpCl. $plan(X)$ plays the role of a classical planner that returns a set of solutions of X. We assume that $plan(X)$ returns one solution at a time, **nil** if there is no more solution, or **failed** if X does not have a solution. $is\_solution(\beta, P)$ checks whether or not β is a solution of the problem P.

Algorithm 1 CpCl(P)

```
Input: A planning problem $P = \langle F, O, I, G\rangle$
Output: A solution for $P$
Let $\Sigma = [s_0, \ldots, s_n] = ext(I)$        {Compute $ext(I)$}
$\alpha_{s_0} = plan(P(s_0))$                {Get a solution of $P(s_0)$}
if $\alpha_{s_0}$ = failed then return failed
while $\alpha_{s_0} \neq$ nil do
    if $is\_solution(\alpha_{s_0}, P)$ then return $\alpha_{s_0}$
    else $\beta = completion(\alpha_{s_0}, P, \Sigma, 1)$
        if $is\_solution(\beta, P)$ then return $\beta$
    $\alpha_{s_0} = plan(P(s_0))$
end while
return unknown
```

$completion(\alpha, P, \Sigma, j)$ ($0 \leq j \leq n$) takes a problem P, whose initial belief state is $[s_0, \ldots, s_n]$, and a solution $\alpha_{s_{j-1}}$ of $P(s_{j-1})$ (where $s_{-1} = s_n$), and attempts to create solutions α_{s_i} for $P(s_i)$, $i = j+1, \ldots, n$.

The procedure constructs a solution of the sub-problem $P(s_i)$ from the solution $\alpha_{s_{i-1}}$ of $P(s_{i-1})$, by inserting actions into $\alpha_{s_{i-1}}$. To achieve this, the procedure starts with the state s_i and an empty plan, considers each action a in $\alpha_{s_{i-1}}$, and executes the following tasks:

- **Task 1**: inserts a sequence of actions before a, so that *(1)* a is executable and *(2)* the execution of a maintains some effects of a. If this fails then the algorithm stops and declares that the original plan cannot be extended to a plan for s_i.
- **Task 2**: makes sure that the final sequence of actions α_{s_i} achieves the goal of $P(s_i)$ and this may require adding extra actions at the end of α_{s_i}.

Implementation and Evaluation

We develop CpCl using the source code of LAMA, the winner of the deterministic track of IPC 2008 because of its exceptional performance and its object-oriented implementation which allows for an easy instantiation a new planning module with different initial state and goal. In order to test CpCl on the wide range of collected conformant planning problems and achieve good performance, we have made the following modifications to LAMA:

- The parser has been modified to consider various types of actions that were rejected by LAMA;
- The parser has also been modified to enable the computation of the initial belief state of the planning problem;

[2] We say that α is a subsequence of β is α is obtained by removing any number of elements from β.

[3] $\circ$ denotes concatenation of two lists.

- Algorithms 1 have been integrated to LAMA. To generate more than one solution of a problem, we disable the A* search feature of LAMA by keeping the open list (queue of unexplored nodes) after the first solution is found and continue the search for the next solution if needed.

We compare CPCL with other state-of-the-art planners—i.e., CPA (Tran *et al.* 2009), DNF (To *et al.* 2009), and t0 (Palacios and Geffner 2009)—on problems from the literature and previous planning competitions. The experiment have been performed on a Core 2 2.66GHz machine, with 4Gb memory, with a run-time cutoff of 30 minutes.

The benchmark set contains 731 instances of 18 domains from the recent IPCs (2006 and 2008) and from the distribution of CFF and t0. CPCL is able to solve 684 instances while other planners can only solve up to 417 instances(333, 360 and 417 for CPA(H), t0 and DNF respectively).

Instance	CPA(H)	t0	DNF	CPCL
blw-03	20.4/205	48.51/80	307/325	**1.3**/266
blw-04	AB	AB	AB	**29.5**/1384
coins-10	**0.03**/48	0.04/26	0.20/27	0.037/36
coins-30	AB	AB	AB	**1.0**/1107
comm-15	2.29//95	**0.092**/110	3.43/125	0.1/97
comm-25	1222/389	1.55/453	1797/501	**0.8**/294
sortnet-5	**0.02**/13	0.18/15	0.03/15	0.05/15
sortnet-15	240/74	AB	**35**/118	63.9/120
sortnum-5	AB	1.9/10	1.67/10	**0.81**/10
sortnum-20	AB	AB	AB	**12.3**/190
uts-30	4.9/74	0.79/67	1.39/73	**0.17**/64
uts-cycle-03	**0.01**/3	0.14/3	0.01/3	0.04/3
uts-cycle-15	AB	AB	AB	**1314**/272
raos-keys-02	0.26/32	**0.02**/21	0.09/39	0.05/38
raos-keys-04	AB	AB	AB	**16.78**/163
forest-03	AB	0.62/45	TO	**0.46**/167
forest-09	AB	AB	AB	**183.8**/963

Table 1: Results for IPC 2006/08 Domains (Time in *seconds*)

Domains from IPCs: Table 1 contains the results of our experiments with domains from the IPCs 2006 and 2008—in terms of the time and length of the first solution reported by each planner. Boldface indicates the fastest planner. AB denotes an execution aborted by the planner due to"out of memory," and TO denotes time-out. CPCL performs exceptionally well, both in term of efficiency and scalability. CPCL consistently outperforms other planners in large instances. For space reason, we omit the result on domains from the distribution of CFF and t0.

Conclusion

In this paper, we proposed a novel approach to conformant planning using classical planner. We implemented the new algorithm using the source code of the classical planner LAMA, and evaluated the new planner, CPCL, against state-of-the-art conformant planners. CPCL outperforms other planners in both performance and scalability, indicating that the proposed approach is a strong alternative to current state-of-the-art approaches.

It is well-known that the scalability and performance of a conformant planner depend on two factors: the ability to deal with the potential large size of the initial belief state and the ability to guide the search. CPCL deals with these two problems by taking advantage of the best from the research in conformant and in classical planning.

CPCL copes with the huge size of the initial belief state by considering each possible initial state *separately*, which reduces the memory requirements—which is often the problem for other planners. This also allows CPCL to easily take advantage of techniques that have been developed in conformant planning research for reducing the size of the initial belief state. By converting the problem to a classical problem, CPCL can exploit the best heuristic classical planners in computing a solution. Furthermore, the generate-and-complete algorithm allows CPCL to generate a solution by solving multiple smaller problems.

We observe that, even though CPCL can solve a wide range of benchmarks from various sources, which seem to be difficult for other state-of-the-art conformant planners, there are still domains in which CPCL does not work well. Among them, the adder domain seems to be the most difficult one. This domain is special in that the size of the initial belief state is very small, but the number of actions which can be executed in a state is very large. Furthermore, the conditional effects of the actions are much more complex than those in other domains. We hypothesize that these two factors make this domain difficult for conformant planners.

References

D. Bryce and O. Buffet. The uncertainty part of the 6th IPC, 2008.

R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. In *ICAPS 2004*, pages 355–364, 2004.

H. Palacios and H. Geffner. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes). In *AAAI*, 2006.

H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *JAIR*, 35:623–675, 2009.

S. Richter and M. Westphal. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *JAIR*, 39:127–177, 2010.

D.E. Smith and D.S. Weld. Conformant graphplan. In *AAAI*, pages 889–896, 1998.

S. T. To, E. Pontelli, and T. C. Son. A conformant planner with explicit disjunctive representation of belief states. In *ICAPS 2009*, AAAI, 2009.

D. V. Tran, H. K. Nguyen, E. Pontelli, and T. C. Son. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *PADL 2009*, pages 239–253. Springer, 2009.

S.W. Yoon, A. Fern, and R. Givan. FF-Replan: A baseline for probabilistic planning. *ICAPS*, 352–259. AAAI. 2007.

Automatic Polytime Reductions of NP Problems into a Fragment of STRIPS

Aldo Porco
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
aldo@gia.usb.ve

Alejandro Machado
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
alejandro@gia.usb.ve

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Abstract

We present a software tool that is able to automatically translate a given NP problem into a STRIPS problem such that the former problem has a solution iff the latter has one, a solution for the latter can be transformed into a solution for the former, and all this can be done efficiently. Moreover, the tool is built such that it only produces problems that belong to a fragment of STRIPS that is solvable in non-deterministic polynomial time, a fact that guarantees that the whole approach is not an overkill (from the perspective of complexity theory). This tool has interesting applications. For example, with the advancement of planning technology, it can be used as an off-the-shelf method to solve general NP problems with the help of planners, to automatically generate benchmark problems of known complexity in a systematic and controlled manner, and to understand the main deficiencies of the heuristics or search algorithms used in planning. More interesting however is the relevance of the approach for the area of Knowledge Engineering in which one of the goals is to devise automatic methods for obtaining planning problems from declarative descriptions of real-world tasks, and for the field of Descriptive Complexity Theory on which the approach rests.

(*The full paper on this system appear in the ICAPS-11 Proceedings at pages 178-185*)

A Visual Programming Interface for Specifying Plan Dynamics (Demo Paper)

Julie Porteous, Jonathan Teutenberg, David Pizzi, Fred Charles and **Marc Cavazza**
School of Computing,
Teesside University,
Middlesbrough TS1 3BA,
United Kingdom
{j.porteous,j.teutenberg,d.pizzi,f.charles,m.o.cavazza}@tees.ac.uk

Abstract

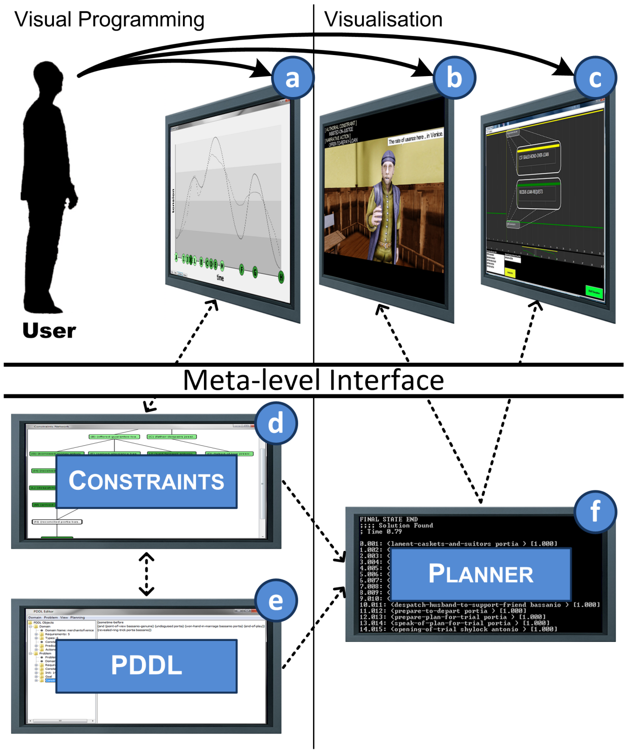
When planning techniques are used for narrative generation in new media applications different criteria, based on such things as plan dynamics, are required to assess plan quality. In our work we have looked at providing support for specifying this information: we introduced a meta-level of representation that is an abstraction of the domain with respect to time and causality which we have represented visually via a *narrative arc*. We have used this visual representation in a visual programming approach to the exploration and specification of plan dynamics. In this demo we showcase this approach using our system that features virtual characters inspired by Shakespeare's play *The Merchant of Venice*[1].

Introduction

Interactive Storytelling (IS) represents a major new application area for AI planning. New media domains such as IS differ markedly from the benchmark domains that have featured in AI research. One key difference is that the criteria used to assess plan quality are no longer concerned with such things as optimality, rather the focus is the dynamics of the plan, in terms of the shape of its *trajectory* and the intermediate states that will be traversed when it is executed.

In earlier work we developed a plan-based approach to narrative generation that exploits a meta-level of representation via the use of constrained predicates, referred to as *constraints*, representing key narrative situations for the domain of interest (Porteous, Cavazza, and Charles 2010). These constraints are used as intermediate goals to guide generation of narratives featuring these situations. Hence they provide a way to specify the abstract shape of a narrative trajectory in terms of the intermediate states that it will traverse when it is visualised. This abstract meta-level of representation formed the basis for our solution to the problem of specifying plan dynamics: we developed a visual representation in the shape of a narrative arc which facilitated a visual programming approach to specification of plan dynamics.

In the demo we showcase our approach to specifying plan dynamics with reference to an IS system we have developed based on Shakespeares' *Merchant of Venice*. The demo is a companion to our ICAPS paper (Porteous et al. 2011).



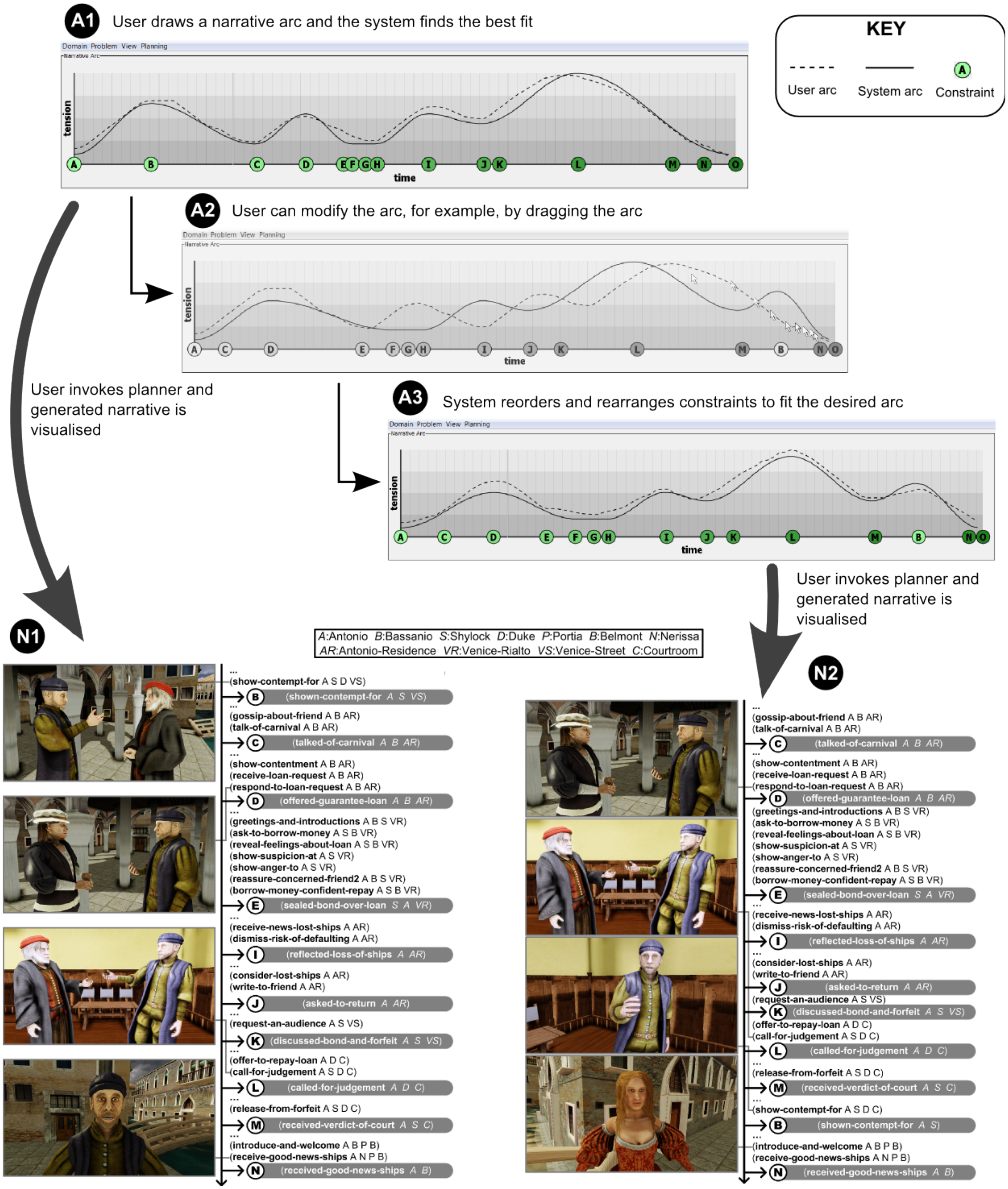
```
FINAL STATE END
;;;; Solution Found
; Time 0.79

0.001: <lament-caskets-and-suitors portia > [1.000]
1.002: <
2.003: <
3.004: <
4.005: <
5.006: <
6.007: <
7.008: <
8.009: <
9.010: <
10.011: <despatch-husband-to-support-friend bassanio > [1.000]
11.012: <prepare-to-depart portia > [1.000]
12.013: <prepare-plan-for-trial portia > [1.000]
13.014: <speak-of-plan-for-trial portia > [1.000]
14.015: <opening-of-trial shylock antonio > [1.000]
```

Figure 1: System Architecture Overview. User Interaction is at the meta-level via visual programming (a), visualisation (b) and timeline (c) and not with lower level components.

Demo: System Architecture

Our visual interface for specification of plan dynamics is fully implemented and integrated with our IS system. An overview of the system architecture is shown in figure 1. The user interacts at the meta-level via a *Narrative Arc Window* (a) and can also explore generated narratives via two visualisation windows: an *Animation Window* (b); and a *Timeline Window* (c). The system also features some hierarchically organised lower level components. They include the constraints (d) and other PDDL constituents of the domain model (e). The planner is invoked by the user (f) to explore the narrative possibilities of different plan dynamics.

[1] The full paper describing this system appears in the ICAPS-11 Proceedings at pages 186-193


A1
User draws a narrative arc and the system finds the best fit
Domain Problem View Planning
Narrative Arc
tension
time
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
KEY
User arc
System arc
Constraint
A2
User can modify the arc, for example, by dragging the arc
A3
System reorders and rearranges constraints to fit the desired arc
User invokes planner and generated narrative is visualised
User invokes planner and generated narrative is visualised
N1
N2
A:Antonio B:Bassanio S:Shylock D:Duke P:Portia B:Belmont N:Nerissa
AR:Antonio-Residence VR:Venice-Rialto VS:Venice-Street C:Courtroom

N1

```
...
(show-contempt-for A S D VS)
→ B (shown-contempt-for A S VS)
...
(gossip-about-friend A B AR)
(talk-of-carnival A B AR)
→ C (talked-of-carnival A B AR)
...
(show-contentment A B AR)
(receive-loan-request A B AR)
(respond-to-loan-request A B AR)
→ D (offered-guarantee-loan A B AR)
...
(greetings-and-introductions A B S VR)
(ask-to-borrow-money A S B VR)
(reveal-feelings-about-loan A S B VR)
(show-suspicion-at A S VR)
(show-anger-to A S VR)
(reassure-concerned-friend2 A B S VR)
(borrow-money-confident-repay A S B VR)
→ E (sealed-bond-over-loan S A VR)
...
(receive-news-lost-ships A AR)
(dismiss-risk-of-defaulting A AR)
→ I (reflected-loss-of-ships A AR)
...
(consider-lost-ships A AR)
(write-to-friend A AR)
→ J (asked-to-return A AR)
...
(request-an-audience A S VS)
→ K (discussed-bond-and-forfeit A S VS)
...
(offer-to-repay-loan A D C)
(call-for-judgement A S D C)
→ L (called-for-judgement A D C)
...
(release-from-forfeit A S D C)
→ M (received-verdict-of-court A S C)
...
(introduce-and-welcome A B P B)
(receive-good-news-ships A N P B)
→ N (received-good-news-ships A B)
```

N2

```
...
(gossip-about-friend A B AR)
(talk-of-carnival A B AR)
→ C (talked-of-carnival A B AR)
...
(show-contentment A B AR)
(receive-loan-request A B AR)
(respond-to-loan-request A B AR)
→ D (offered-guarantee-loan A B AR)
(greetings-and-introductions A B S VR)
(ask-to-borrow-money A S B VR)
(reveal-feelings-about-loan A S B VR)
(show-suspicion-at A S VR)
(show-anger-to A S VR)
(reassure-concerned-friend2 A B S VR)
(borrow-money-confident-repay A S B VR)
→ E (sealed-bond-over-loan S A VR)
...
(receive-news-lost-ships A AR)
(dismiss-risk-of-defaulting A AR)
→ I (reflected-loss-of-ships A AR)
...
(consider-lost-ships A AR)
(write-to-friend A AR)
→ J (asked-to-return A AR)
(request-an-audience A S VS)
→ K (discussed-bond-and-forfeit A S VS)
(offer-to-repay-loan A D C)
(call-for-judgement A S D C)
→ L (called-for-judgement A D C)
...
(release-from-forfeit A S D C)
→ M (received-verdict-of-court A S C)
...
(show-contempt-for A S D C)
→ B (shown-contempt-for A S)
...
(introduce-and-welcome A B P B)
(receive-good-news-ships A N P B)
→ N (received-good-news-ships A B)
```

Figure 2: An overview of visual programming of plan dynamics. Screenshots A1, A2 and A3 show an interactive process where an initial user drawn narrative arc (A1) is modified by dragging different sections of the arc (A2) and where the system arc and position of constraints has been automatically adjusted to reflect the user changes (A3). N1 and N2 are representations of the narratives that have been generated in response to user invocation of the planner using the arcs in A1 and A3 respectively.

Demo: Sample User Session

The aim of the demo is to show how user specification of different shaped narrative arcs results in the generation of different narratives. In the demo we will also show these generated narratives visualised in a 3D world and demonstrate tools which support user evaluation of them. Here we illustrate highlights of this process via an example scenario.

A user will typically start by interacting with the visual programming system via the Narrative Arc Window. This enables them to draw and manipulate differently shaped narrative arcs. At any point the user can use this narrative arc to drive narrative generation and to explore the visualisation of this narrative in a 3D world. For the user this means that once a narrative arc has been created they can choose to invoke the narrative generator and the system will begin visualisation of the narrative in a visualisation window. Figure 2 gives an illustration of some important aspects of this process of user interaction with the system.

For instance, figure 2 includes a series of screen shots that show user interaction via the narrative arc (A1, A2 and A3). They show the constituent elements of the window that are presented to the user: the x-axis is the duration of the narrative; the y-axis is the level of narrative tension; and the labelled circles along the x-axis represent narrative constraints (when the user runs the mouse over one of the circles then the constraint name is displayed). Screenshot A1 shows a user drawn desired narrative arc along with an arc that has been generated by the system which is its best fit to the users arc (further detail of this matching process can be found in (Porteous et al. 2011)). Screenshot A2 shows the user in the process of manipulating their desired arc in order to specify different plan dynamics: the user is modifying the arc by dragging different segments of it. In response to the user modifications the system recalculates the best fit and redisplays the constraints and system arc, which results in the situation shown in screenshot A3. For the constraints along the x-axis, observe how both the relative spacing between them and the ordering of the constraints has changed. In particular, constraint B has moved from its position between constraints M and N, to between A and C.

Once a user has specified the shape of their desired narrative arc, they can then choose to generate a narrative that displays those global properties. As an illustration the narratives that have been generated using the arcs in screenshot A1 and A3 are represented in narratives N1 and N2 respectively. The shapes of the arcs in A1 and A3 are very different: for instance, the arc A1 follows an Aristotelian contour, with minor climaxes of increasing tension levels before the final climax of the play and subsequent denouement. This climax is the end of the "pound-of-flesh" sub-plot (Hinely 1980), where it appears all hope is lost for the titular merchant of the play, Antonio, having defaulted on a loan from Shylock, who is unwilling to show mercy and finally begs the court to deliver its judgement represented in our domain with constraint L (called-for-judgement antonio duke courtroom). Segments of the narrative generated from the arc A1, along with shots from its 3D visualisation, are shown in narrative N1. The constraints labelled M and B are interesting in this narrative because they show Antonio revealing his true feelings of contempt for Shylock only after Shylock has continued to demand his pound-of-flesh and failed to show mercy on Antonio. One consequence of these actions is that some justification is given for Antonio's demonstration of contempt for Shylock.

In contrast, arc A3 in figure 2 has a different shape with early and late climaxes with minor crises through the middle section. This user arc has resulted in a different arrangement of constraints and system arc (to that discussed already for arc A1) and the semantics of the resulting narrative (shown in N2) are very different since the constraint that relates to Antonio's bad treatment of Shylock, constraint B, now appears at the start of the narrative. This illustrates how different narrative arcs and the plan dynamics they specify can reveal different semantics. For the generated narrative N2, the semantics are very different to those of narrative N1 since Antonio's merciless treatment of Shylock early on has no justification and any later suffering by Antonio can be seen as deserved.

Our system also features a timeline window, as shown in the system architecture figure on page 3. In this window the segment of the narrative that is currently being visualised is displayed to the user. As the planner proceeds the timeline is updated in real-time. Users can use the timeline for narrative navigation, for example, to rewind and restart the visualisation from a different point in the narrative.

Acknowledgements

This work has been funded (in part) by the European Commission under grant agreement IRIS (FP7-ICT-231824).

We would like to thank Terry Borst for drawing our attention to the important role of trajectory representation in the authoring of interactive narratives.

References

Hinely, J. L. 1980. Bond Priorities in The Merchant of Venice. *Studies in English Literature, 1500-1900* 20(2):217–239.

Porteous, J.; Teutenberg, J.; Pizzi, D.; and Cavazza, M. 2011. Visual Programming of Plan Dynamics using Constraints and Landmarks. In *Proc. of the 21st Int. Conf. on Automated Planning and Scheduling (ICAPS 2011).*

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying Planning to Interactive Storytelling: Narrative Control using State Constraints. *ACM Transactions on Intelligent Systems and Technology (ACM TIST)* 1(2):1–21.

Beyond Calendar Mashups: SELFPLANNER 2.0

Ioannis Refanidis[1], Anastasios Alexiadis[1] and Neil Yorke-Smith[2]

[1] University of Macedonia, Dept. of Applied Informatics, Thessaloniki, Greece
{yrefanid, talex}@uom.gr
[2] American University of Beirut, Lebanon and SRI International, US
nysmith@aub.edu.lb

Abstract

Modern electronic calendars offer a variety of functionalities to help a user organize her activities—her tasks and events. However, rarely do these tools support automated scheduling and rescheduling of a user's activities. This demo paper presents SELFPLANNER 2.0, the latest version of a web-based calendar prototype that helps a user to organize her activities by coupling a rich activity model with a scheduling engine. Activities are considered as having temporal domains, utilities, variable durations, and alternative locations; they may be interruptible or periodic; and they may be concurrent. The user is able to express constraints and preferences over the way individual activities or pairs of activities are scheduled. The underlying scheduler seeks to maximize the overall schedule utility using a greedy approach. SELFPLANNER employs Google Calendar for presentation and Google Maps to compute travelling times for temporally adjacent activities scheduled in distant locations.

Introduction

Paper calendars have, for more and more people, given way to electronic calendaring tools. Web-based calendar applications, such as Google Calendar, Yahoo! Calendar, or 30 Boxes, allow the user to connect from a variety of devices without the need for synchronization. They offer intuitive functionality, such as reminders, multiple overlapping calendars, manual meeting arrangement, calendar sharing and publishing, to-do task lists, and so forth. Similar functionalities are provided by client-side calendar applications such as Apple's iCal and Microsoft's Outlook.

Whether web-based or client-based, no popular calendaring applications provide the means for effective automated activity scheduling. By automated scheduling we refer to the use of a scheduler that decides where to put an activity—an event or a task—within a user's calendar, subject to the user's approval. In order for a scheduler to decide when to place an activity, a rich formulation of the scheduling problem is necessary: otherwise the proposed schedule would be rejected by the user as unrealistic. This problem formulation should specify, for each activity: the allowed time intervals (that is, its temporal domain), its duration and whether the duration is fixed or variable, whether the activity is interruptible or not, whether it is periodic or not, where should the user be in order to accomplish it (e.g., the location of the appointment), and how much time is needed in order for the user to move from one location to another. Further, the formulation should reflect the user's preferences, that is the utility gained when an activity is accomplished, as well as additional utility gained by the way an activity is scheduled within its domain (for example, earlier in the day may be preferred to later in the day), and any utility gained by the way pairs of activities are scheduled in relation to each other.

In this demo paper we present SELFPLANNER, a web-based prototype calendar application that couples such a rich activity model with a scheduling engine. Online since 2008, SELFPLANNER has several dozen real users. An earlier version of the system, circa 2008, was evaluated by its users with very promising results (Refanidis and Alexiadis, 2011). The latest version of the system, version 2.0, illustrated in this paper, implements a significantly expanded model (Refanidis and Yorke-Smith, 2010). SELFPLANNER uses Google Calendar for presentation purposes and Google Maps to compute travelling times. The system is not designed to support automated meeting arrangement; the user can use the manual meeting arrangement tools provided by Google Calendar, with the time of the meeting being considered as busy time by SELFPLANNER when scheduling the rest of the activities. The system is accessible through http://selfplanner.uom.gr.

This demo paper illustrates the complexity that an intelligent calendar assistant must address, through a set of motivating examples. We then present the key features of the system, and conclude by anticipating future developments in intelligent calendar applications.

Motivating Examples

The vision behind SELFPLANNER is to consider both events and tasks when providing automated scheduling assistance to support the user's time management. Traditionally, events such as a doctor's appointment are placed in a user's calendar, with a specific time (and potentially location) reference, whereas tasks sit in a to-do list, being characterized perhaps by a deadline. Treating events and tasks differently, with only events having their place into a user's calendar, can result in tasks missing their deadlines. Apple iCal and Microsoft Outlook allow the user to drop a task from the to-do list into the calendar, thus allocating manually some time slots to the task. Google's applications weakly integrate tasks, events, and email; a strong integration is adopted by the open source Chandler Project. With SELFPLANNER, any activity that requires some of the user's time has its place in her calendar. In the following paragraphs we give some motivating examples.

The simplest case of an activity is one that has a designated schedule. For example, attending a lecture. The user has only to decide whether or not to attend the lecture. If she decides to attend, there is no option to negotiate the time when or the location where the lecture will take place. By contrast, activities such as buying food are more flexible: the user can decide between alternative schedules. Consequently, providing automated assistance in scheduling such activities relies on a wealth of information (Krzywicki et. al., 2010), such as:

- A duration estimate (how long?)
- The timetable of the shops (when are they open?)
- The alternative shops which are compatible with the shopping list (which shops sell food?)

Going shopping can also be considered a periodic activity, i.e., one that is repeated, say once a week. While periodic activities do not have to be scheduled in the same time and location within each period, they should be scheduled once within each period, whenever possible. For example, going shopping may take place either on Friday or on Saturday, either in the morning or in the evening, depending on the rest of the activities in the user's calendar within each week. The location for each instance of a periodic activity can also be selected based on the locations of other activities that are scheduled adjacently in time, with the aim to minimize travelling time.

Some activities, such as reading a book, are not usually performed all at once: we call these activities interruptible. The user might estimate an overall duration for this activity, e.g., 30 hours, and the activity will be divided into parts that will be scheduled separately. These parts do not have to share the same duration or location, but their sum should equal the overall duration specified for the activity. The user might specify compatible locations where she should be in order to execute the activity and, potentially, a temporal domain. Note that periodicity and interruptibility are orthogonal properties: an activity can be periodic or interruptible, or both.

Activities may serve as placeholders or reminders (Palen, 1995). It is a common situation to have overlapping activities in a user's calendar. For example, one might define a three-day activity concerning attending ICAPS 2012 in Brazil and several other more specific activities concerning attending specific sessions of the conference.

Busy users often have an overloaded schedule, such that all of their activities cannot fit within their calendars or they cannot all be scheduled in an ideal way. Some decisions have to be taken concerning which activities to schedule and how. Each activity included in the user's calendar presumably yields some utility to the user, which depends on the activity. For instance, doing business may be considered more important by some than spending time for leisure activities. However, whether or not an activity is included in a user's calendar is not the only source of utility. Utility may also result from the way an activity is scheduled, both on its own as well as in conjunction with the rest of them. Hence, it might be preferable to decide not to schedule an activity at all, in order to get a better schedule for the rest of them.

As an illustration, consider sleeping as a typical example of an activity for which alternative schedules result in different degrees of user's satisfaction. The user might prefer to 'execute' this activity at once within each day than executing it in parts. Usually the user prefers to schedule this activity during the night than during the day. She also might prefer to schedule this activity at home than at her office! Finally, sleeping has a variable duration, in the sense that the user can decide how long to sleep (we do not consider stochastic activities, where duration is a random variable). For example, one might want to sleep between 6 and 8 hours daily, with 8 hours resulting in more utility than 6 hours.

Further, utility may also result from the way sets of activities are scheduled in relation to each other. Consider writing a paper and doing housekeeping. Writing the paper is a heavy mental task that is optimized when one has clear head and is concentrated on it. One might prefer to work on writing the paper before doing housekeeping or, alternatively, to leave some minimum temporal distance between the two activities. However, in case it is not possible to schedule these activities in the optimum way, the user is willing to accept any schedule that includes these activities, provided she completes both, even if the schedule violates some of her ideal preferences.

Finally, and not least, constraints can hold between activities. For example, consider a pair of activities concerning going to Crete on holiday by plane, and the return journey. Obviously, going to Crete should precede the return flight (an ordering constraint), whereas we could

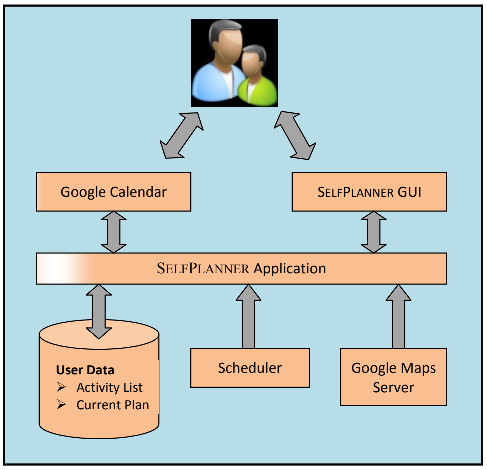


Figure 1. SELFPLANNER architecture.

ask for a minimum and maximum duration for the whole journey (a temporal proximity constraint between these two activities). Other activities to be executed during the vacation should be scheduled between the two flights. Implication constraints might also hold between these activities, imposing for example that there is no reason to schedule one of the two flights—or any other of the activities that are related with the vacation—without scheduling the other.

Scheduling a user's activities is a continuous process. New activities might arrive at any time, causing already scheduled activities to be rescheduled or even abandoned in order to accommodate the new ones within the user's calendar. The need for rescheduling is critical for any intelligent calendar application. In order to minimize the user's disturbance, rescheduling should give less preference in changing the user's short-term schedule.

The System

SELFPLANNER 2.0 supports all the modelling aspects one needs in order to represent the situations described with the motivating examples, and many more. Activities are characterized by a utility value, a duration range (with greater duration resulting in greater utility), a temporal domain, a preference function over the various time slots of the temporal domain, a set of alternative locations, and a utilization parameter, concerning the percentage of a user's attention required when the activity is executed, with the constraint that the sum of the utilizations of all concurrent activities does not exceed 100%. In order for a set of activities to be scheduled concurrently, they need to have the same (or compatible) locations. Temporal preference functions supported are linear descending (the earliest the better), linear ascending (the latest the better), stepwise descending (before some specified time point), and stepwise ascending (after some specified time point). The user is free to specify as little or as much of the information for an activity as she likes.

For interruptible activities the user can specify the minimum and maximum allowable duration for their parts. Further, the user can specify extra constraints—either mandatory hard constraints or flexible soft constraints—concerning the minimum and the maximum temporal distance between pairs of parts of the interruptible activity. Soft constraints, if satisfied, result in extra utility; partial satisfaction is supported. Four types of binary hard and soft constraints are supported: Ordering constraints, minimum distance constraints, maximum distance constraints, and implication constraints. An activity can also be periodic, with each instance of the activity being treated as a separate activity by the scheduler. Daily, weekly and monthly periodic activities are supported.

The scheduling problem that results can be framed as a constraint optimization problem. SELFPLANNER uses a greedy heuristic scheduling engine, based on the Squeaky Wheel Optimization framework, boosted with domain-dependent heuristics, to solve the underlying constraint optimization problem. Experimental results have shown that even problems with dozens of fully featured activities can be solved near optimally in a few seconds (Refanidis and Yorke-Smith, 2010; Refanidis, 2007).

The system's architecture is shown in Figure 1. On the server side, the SELFPLANNER application implements the system's logic; a scheduling engine generates plans for the user's activities; and a database retains users' data, including their activity lists and current plans. The user interface consists of a Java applet running on the client side. The user interacts also with Google Calendar, in order to watch her plan, which is updated by the main SELFPLANNER application. The user can also add new events directly into her Google Calendar, e.g., meetings with other people; these events are considered busy time by the system when scheduling her activities. Finally, the SELFPLANNER application interacts with the Google Maps Server, in order to compute travelling times between the locations of the activities.

Special attention is given to the user interface. While many parameters may be specified for each activity, most of them usually have default values. The most important features of the system (Figure 2) are the following:

- Users indicated that the most time-consuming task concerns the definition of a temporal domain. SELFPLANNER implements an innovative way to define temporal domains, which consists of a combination of template application and manual editing. A template is a user defined pattern of time slot inclusion/exclusion covering a day, a week, or a month. A template can be applied over the entire temporal domain of an activity, or over a part of it, to include or exclude specific time

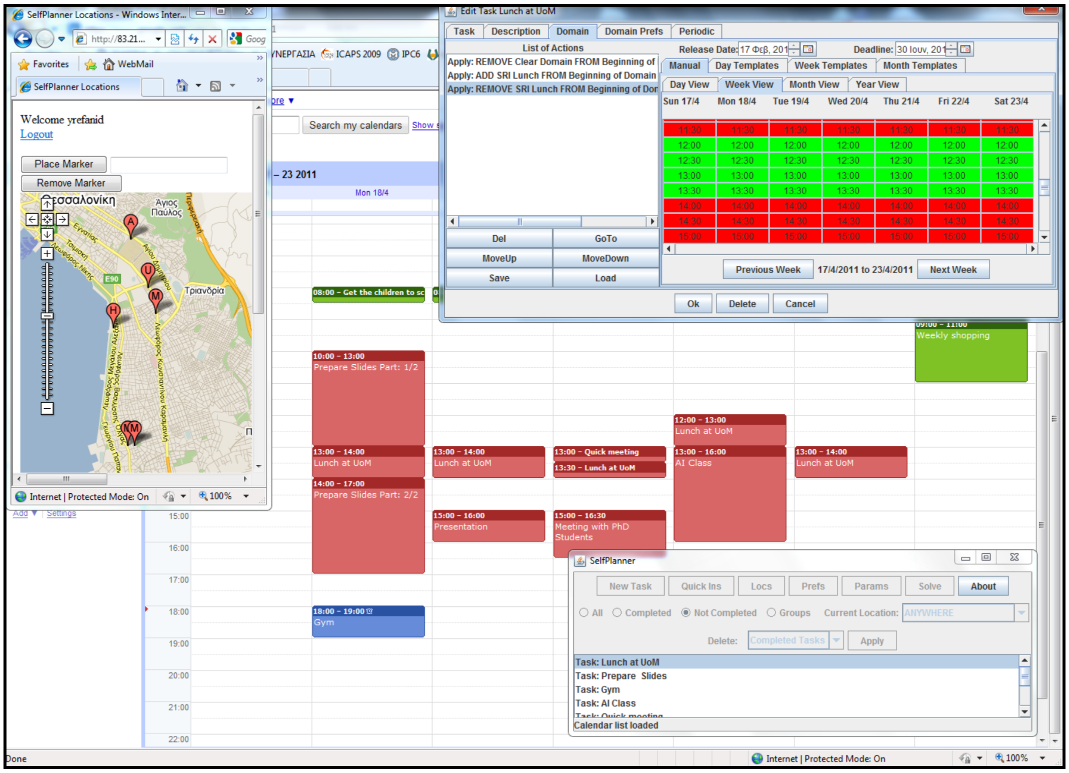


Figure 2. An overview of the SELFPLANNER system. The main application window (lower right) contains the current list of activities. In the upper right corner is the Edit Task dialog box, shown editing the temporal domain of the Lunch at UoM activity. It is interesting to have a look at how this activity has been scheduled. It is a daily periodic activity with a duration that may be either 30 mins or 1 hour (preferred). The restaurant is open from 12:00 to 14:00 daily, and the user prefers to eat as late as possible, i.e., close to 14:00, more strongly than she prefers to have 1 hour for lunch. Seen on the left is the interface for designating relevant locations. Behind the other windows is the Google Calendar interface in which the whole schedule is shown.

slots. Sequences of templates can also be applied. Finally, the user can edit the temporal domain manually. Using templates, the user can define large temporal domains with a few clicks.

- Daylight savings time is taken into account when scheduling, whereas when displaying the resulting plan, the system takes into account the user's timezone. Multiple calendars are supported.
- User defined classes of locations, e.g., malls, are also supported, giving the user the option to assign to an activity large sets of locations with a single choice.
- Specific instances of a periodic activity can be excluded from a user's calendar easily, without the need to resort to the calendar.
- Each time the user logs into the system, she is offered the opportunity to give feedback concerning the status of any activity that should have been accomplished between her last logout and the current time. Hence, the system knows which already scheduled activities were not accomplished and reschedules them.
- The user is able to specify her current location (with the default current location being the location of the last accomplished activity). Hence, there is always enough time to travel to the location of the next activity.
- Activities directly entered into the user's Google Calendar are considered as busy time, so no other activity is scheduled in the same time period. However, these activities are not annotated with a location, so travelling time cannot be computed.
- Since many activities are inflexible, there is an option for rapidly inserting a new non-interruptible activity with specific start time and location. This is equivalent with directly entering the activity into the user's Google Calendar, however doing it through SELFPLANNER allows to specify the activity's location.

Conclusions and Future Work

SELFPLANNER is a continuously evolving prototype application aimed at reducing the effort in the management of a user's electronic calendar. Currently it implements a rich activity model with simple, interruptible and periodic activities, locations, concurrency, and several unary and binary hard and soft constraints, coupled with a powerful best-effort heuristic scheduling engine. An evaluation of an earlier version of the system, since 2008, with real users, demonstrated its strengths over traditional electronic calendars as well its potential for broader user adoption (Refanidis and Alexiadis, 2011).

Currently, SelfPlanner is being evolved through its integration in an information system supporting the visitor of an area to enhance his visit with cultural activities. The integrated system, called myVisitPlanner, will retain information about cultural activities offered in the area of northern Greece (particularly Macedonia and Thrace), including metadata such as location, working hours, estimated visit duration etc. Coupling this information with the user's calendar and profile, the system will be able to suggest activities to the user and, once accepted, schedule them in his calendar, while taking into account constraints imposed both by the new as well as the existing activities.

SelfPlanner is a step towards creation of general purpose intelligent user assistants. Such assistant agents aim to helping the user to organize and accomplish her tasks (Maes, 1994; Freed et. al. 2008). One aspect of their duties is to help organize the user's time, i.e., advising what to do and when, or supporting calendar management and meeting coordination (Berry et. al. 2011). However, there are other ways in which SelfPlanner could be extended in order to provide true intelligent assistance in organizing a user's activities. For example:

- Coordination with other users: Currently SelfPlanner organizes only the activities of a single user, with meetings with other users being considered as busy time. Assuming that several users are using a system with scheduling capabilities, arranging meetings could involve rescheduling of already scheduled activities for all users (Berry et al. 2011).
- Planning problem: The current activity model could be enhanced with allowing the activities having preconditions and effects. Consequently, the scheduling engine should be replaced by a planning engine, able to add activities into the user's calendar in order to support the open goals.
- Parameter learning: Instead of information about activities being entered by the user, machine learning can be exploited to more intelligent pre-populate fields based on previous activities (Krzywicki et. al. 2010). While the knowledge entry and management has not been a focus of our research, we are aware it is significant for real-life adoption of intelligent agents.
- Information extraction: An intelligent agent should be aware of its user's profile and exploit this knowledge in order to extract information, such as from the internet, concerning activities that might be of interest for its user, e.g., attending ICAPS 2012 (Oh et. al., 2010).
- Web service invocation: Finally, a real intelligent agent should be able to accomplish some activities automatically, be invoking suitable web services if available (Freed et. al. 2008). For example, for an already scheduled activity concerning travelling to Brazil to attend ICAPS 2012, booking the plane tickets and the hotel, or getting information for the weather conditions, might save a lot of the user's time.

Acknowledgements

The first two authors are supported for this work by the European Union and the Greek Ministry of Education, Lifelong Learning and Religions, under the program "Competitiveness and Enterprising" for the areas of Macedonia-Thrace, action "Cooperation 2009", project "A Personalized System to Plan Cultural Paths (myVisitPlanner^{GR})", code 09SYN-62-1129.

ΕΥΡΩΠΑΙΚΗ ΕΝΩΣΗ
ΕΥΡΩΠΑΪΚΟ ΤΑΜΕΙΟ
ΠΕΡΙΦΕΡΕΙΑΚΗΣ ΑΝΑΠΤΥΞΗΣ

ΑΝΤΑΓΩΝΙΣΤΙΚΟΤΗΤΑ
ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΤΗΤΑ
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΠΑΝ II
η περιφέρεια στο επίκεντρο της ανάπτυξης

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη

Υπουργείο Παιδείας, Δια Βίου Μάθησης και Θρησκευμάτων
ΓΓΕΤ – ΕΥΔΕ-ΕΤΑΚ

Ε. Π. Ανταγωνιστικότητα και Επιχειρηματικότητα (ΕΠΑΝ ΙΙ), ΠΕΠ Μακεδονίας – Θράκης, ΠΕΠ Κρήτης και Νήσων Αιγαίου, ΠΕΠ Θεσσαλίας – Στερεάς Ελλάδας – Ηπείρου, ΠΕΠ Αττικής

References

Berry, P.M., Gervasio, M., Peintner B., and Yorke-Smith, N. 2011. PTIME: Personalized Time Management. *ACM Transactions on Intelligent Systems and Technology* (to appear).

Freed, M., Carbonell, J., Gordon, G., Hayes, J., Myers, B., Siewiorek, D., Smith, S.F., Steinfeld, A. and Tomasic, A. 2008. RADAR: A Personal Assistant that Learns to Reduce Email Overload. Proc. of 23rd AAAI Conference on Artificial Intelligence (AAAI-08), Chicago, IL, USA. AAAI Press.

Krzywicki, A., Wobcke, W. and Wong, A., 2010. An Adaptive Calendar Assistant Using Pattern Mining for User Preference Modelling. Proc. of 14th International Conference on Intelligent User Interfaces (IUI'10), Hong Kong, China, February 2010.

Maes, P., 1994. Agents that Reduce Work and Information Overload. *Journal of ACM*, 37(7).

Oh, J., Meneguzzi, F. and Sycara, K.P., 2010. ANTIPA: An Agent Architecture for Intelligent Information Assistance. Proc. of the 19th European Conference on Artificial Intelligence (ECAI-10), Lisbon, Portugal.

Palen, L., 1999. Social, Individual and Technological Issues for Groupware Calendar Systems. Proc. of CHI 99 Conference on Human Factors in Computing Systems, Pittsburgh, PA, USA.

Refanidis, I. 2007. Managing Personal Tasks with Time Constraints and Preferences. Proc. of 17th International Conference on Automated Planning and Scheduling Systems (ICAPS-07), Providence, RI, USA. AAAI Press.

Refanidis, I. and Alexiadis, A. 2011. Deployment and Evaluation of SelfPlanner, an Automated Individual Task Management System. *Computational Intelligence*, 27(1), 41-59.

Refanidis, I. and Yorke-Smith, N. 2010. A Constraint Based Programming Approach to Scheduling an Individual's Activities. ACM Transactions on Intelligent Systems and Technologies, 1(2), 12:1--32.

Planning for Agents with Changing Goals

Kartik Talamadupula[†] and **Paul Schermerhorn**[§] and **J. Benton**[†] and
Subbarao Kambhampati[†] and **Matthias Scheutz**[§]

[†]**Department of Computer Science**
Arizona State University
Tempe, AZ 85287 USA
{krt,rao, j.benton} @ asu.edu

[§]**Cognitive Science Program**
Indiana University
Bloomington, IN 47406 USA
{pscherme,mscheutz} @ indiana.edu

Abstract

One of the most important applications of planning technology is guiding robotic agents in an autonomous fashion through complex problem scenarios. Increasingly, real-world scenarios are evolving in ways that require intensive interaction between human actors and the robotic agent, mediated by a planning system. We propose to demonstrate an integrated system in one such problem that falls under the aegis of an urban search and rescue (USAR) scenario. We show a simulation of a run through one such problem where a mobile robot is given certain goals to achieve in a layout of interest, and discuss how the various capabilities of the planner are instrumental in achieving the agent's goals.

Introduction

One of the earliest motivations of Artificial Intelligence was to provide autonomous control to robotic agents that carry out useful service tasks. Application scenarios for these kinds of tasks span a wide spectrum that includes military drones and mules, household assistance agents and search and rescue robots. The level of autonomy desired of such robotic agents can be achieved only by integrating them with planning systems that can plan not only for the initial goals, but also updates to these goals and the state of the world.

Recent years have seen the emergence of fast planning algorithms and systems that can be used to model a large number of the features that distinguish real world applications from theoretical problem scenarios. Key among these features are time, cost, resources, uncertainty and execution failure. Though a few planners have modeled a subset of these features in the past, the scale-up required to support real world timeframes has only come about recently due to the extensive use of heuristic search methods for plan synthesis. Current planners still impose a number of restrictive assumptions in order to support this scale-up; classical planners are the best example of this. The challenge is to identify the essential features when considering planning support for such real-world scenarios.

In this demonstration, we show one such planning system aiding a robotic agent as it navigates a search and report scenario. We describe the environment that the robot is executing in, as well as its goals and actions, in detail. We then provide a brief overview of the planning system that is integrated with the architecture controlling the robot, and detail some extensions that we had to provide in order to enable successful modeling of the application scenario.

Scenario: Search and Rescue

One of the primary applications of robotic agents is in scenarios where a human actor has a plethora of knowledge about the problem at hand, yet cannot act in the world due to inherent dangers to human life – emergency response and firefighting are among the best examples of such scenarios. In such cases, having a robotic agent as part of the team greatly increases the chances of achieving the desired end-goals (rescuing people, putting out a fire etc.) without exposing the human team-member to risks.

In this demonstration, we consider a specific scenario that we had to provide planning support for to illustrate the challenges that crop up when planning for robots in real-world scenarios. This is the urban search and rescue (USAR) scenario – a team consisting of a human and an autonomous robot is tasked with finding and reporting the location of critical assets (e.g. injured humans) in an urban setting (usually a building). A given USAR task may consist of multiple problems, each with different challenges. The human member of the team usually has intimate knowledge of the setting of the scenario, but cannot perform the required tasks due to inherent dangers like fires, gas leaks, collapsed structures etc. The robot is considered autonomous because the human only communicates with it sparingly, to specify the goals that it must achieve and any information that might be deemed useful for its execution in the world. Examples of tasks in the USAR scenario include transporting essential materials to a specified location or entity; and reconnaissance tasks like reporting the locations of trapped or injured humans to the commander and taking pictures of objects or areas of interest. In the following, we present the specific USAR task that we tested our system on in order to illustrate the inherent planning challenges.

Task: Search and Report

In this problem, the robot's main goal is to deliver essential medical supplies to a specific location within the area of interest – during its run, the robot may be given additional information and goals about other assets. The human team-member (the commander) has intimate knowledge of the building's layout, but is removed from the scene and can only interact with the robot via on-board wireless communication. The robot begins in a long hallway that has doors leading off into rooms on either side. Initially, the robot is unaware that that these rooms may contain injured or trapped humans, and its goal is to reach the end of the hallway to deliver the supplies by a given deadline.

As the robot executes a plan to achieve that goal, the human commander notes that it is passing the rooms and recalls that injured humans may be trapped in these rooms. The commander then passes on this information linking rooms to injured humans to the robot, and specifies a new goal on reporting the location of as many such humans as possible given the time and resource constraints imposed by the achievement of its original goal. In addition, the human commander remembers that rooms have doors, and that these doors must be pushed open in order for the robot to gain access to the room behind that door. The robotic agent (and hence the planner) already has a "push" action encoded as part of its domain theory - this action must be updated in order to reflect the new information from the commander. This requires a change to the world model that the planner is using - the system must process information that is received via natural language (from the commander) and relay it to the system, where update methods are used in order to modify the model.

We demonstrate a run of the robotic agent through the above scenario, while it is supported by the planner. Full details of this demonstration can be found in the attached document that details a storyboard.

Planning System

The planner that we use – *SapaReplan* – is an extension of the metric-temporal planner *Sapa* (Do and Kambhampati 2002) that handles partial satisfaction planning (Benton, Do, and Kambhampati 2009) and replanning (Cushing, Benton, and Kambhampati 2008). Specifically, the planning problem is defined in terms of the initial state, and the set of goals that need to be satisfied. Actions have known (real-valued) costs. Each goal can have a reward and a penalty $\in [0, \infty]$. The reward is accrued when the goal is satisfied in the final state, while the penalty is incurred for not satisfying it. The costs, rewards and penalties are all assumed to be in the same units. The net benefit of a solution plan is defined as the sum of rewards of the goals it achieves, minus the sum of penalties of the goals it fails to achieve, and minus the sum of costs of the actions used in the plan. The use of a reward / penalty model allows our planner to model both opportunities and commitments/constraints in a uniform fashion. A goal with zero penalty is a pure opportunity, while one with zero reward is a pure commitment. A "hard" goal has finite reward but infinite penalty (and thus *must be achieved* by any plan).

The planner consists of three coupled, but distinct parts:

- Search. *SapaReplan* performs a weighted A*, forward search using *net benefit* as the optimization criterion.
- Heuristic. The heuristic used to guide the planner's search is based on well-known relaxed planning graph heuristics where, during search, relaxed solutions are found in polynomial time per state. *Sapa* uses a temporal relaxed planning graph that accounts for the durations of actions when calculating costs and finding relaxed solutions. In the partial satisfaction planning extensions, the heuristic also performs online goal selection. In essence, it solves for all goals (hard and soft) in the relaxed problem and gives a cost for reaching each of them (∞ for unreachable goals). If the cost of reaching a soft goal is greater than its reward, it removes that goal from the heuristic calculation. If the cost of reaching a hard goal is infinity, it marks a state as a dead end. Finally, the difference between the total reward and total cost of the remaining goals is calculated and used as the heuristic value.
- Monitoring / Replanning. The extensions for replanning require the use of an execution monitor, which takes updates from the human-robot team architecture (in this case). Upon receiving an update, the planner updates its knowledge of the "current state" and replans. Replanning itself is posed as a new partial satisfaction planning problem, where the initial and goal states capture the status and commitments of the current plan (Cushing, Benton, and Kambhampati 2008).

Problem Updates New sensory information, goals, or facts given by a human commander can be sent to the planner at any time, either during planning or after a plan has been output. Regardless of the originating source, the monitor listens for updates from a single source from the architecture and correspondingly modifies the planner's representation of the problem. Updates can include new objects, timed events (i.e., an addition or deletion of a fact at a particular time, or a change in a numeric value such as action cost), the addition or modification (on the deadline or reward) of a goal, and a time point to plan from. An example update is given below:

```
(:update
 :objects
    red3 - zone
 :events
    (at 125.0 (not (at red2)))
    (at red3)
    (visited red3)
 :goal (visited red4) [500] - hard
 :now 207.0)
```

All goals are on propositions from the set of boolean fluents in the problem, and there can only be one goal on any given proposition. In the default setting, goals are hard, lack deadlines and have zero reward[1]. All fields in an update specification, with the exception of ":now" (representing the time we expect to begin executing the plan), may be repeated as many times as required, or left out altogether. The intent of allowing such a flexible representation for updates is to provide for accumulation of changes to the world in one place. In the particular example provided, a new object "red3" of type "zone" is declared. In addition, three new *events* are defined, one of them with a temporal annotation

[1] Since these goals are *hard*, they can be seen as carrying an infinite penalty; i.e., failing to achieve even one such goal will result in plan failure.

that describes the time at which that event became true. A new hard goal that carries 500 units of reward is also specified, and the update concludes with the specification of the current time.

As discussed by (Cushing, Benton, and Kambhampati 2008), allowing for updates to the planning problem provides the ability to look at unexpected events in the open world as new information rather than faults to be corrected. In our setup, problem updates cause the monitor process to restart the planner (if it is running) after updating its internal problem representation.

Goal and Knowledge Revision

An important problem that the robot (and planner) must deal with is the specification of the goals that must be achieved in a given task. This goal specification may consist of the actual goals to be achieved, as well as the values of achieving such goals, and priorities and deadlines (if any) associated with these goals. The fact that the system's goals are determined and specified by the human in the loop also introduces the possibility that goals may be specified incompletely or incorrectly at the beginning of the scenario. Such a contingency mandates a need for a method via which goals, and the knowledge that is instrumental in achieving them, can be updated.

The biggest planning challenge when it comes to the problem of goal update and revision is that most state-of-the-art planning systems today assume a "closed world" (Etzioni, Golden, and Weld 1997). Specifically, planning systems expect full knowledge of the initial state, and expect up-front specification of all goals. Adapting them to handle the "open worlds" that are inherent in real-world scenarios presents many challenges. The open world manifests itself in the system's incomplete knowledge of the problem at hand; for example, in the search and report scenario, neither the human nor the robot know where the injured humans may be. Thus an immediate ramification of the open world is that goals may often be conditioned on particular facts whose truth values may be unknown at the initial state. For example, the most critical goal in the USAR scenario – reporting the location of injured humans – is conditioned on finding injured humans in the first place. In this section, we describe recent work on bridging the open nature of the world with the closed world representation of the planner that has been done in the context of the USAR problem.

Open World Quantified Goals

Open world quantified goals (OWQG) (Talamadupula et al. 2010) combine information about objects that *may be* discovered during execution with partial satisfaction aspects of the problem. Using an OWQG, the domain expert can furnish details about what new objects may be encountered through sensing and include goals that relate directly to the sensed objects. Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. Formally, an open world quantified goal (OWQG) is a tuple $\mathcal{Q} = \langle F, \mathcal{S}, \mathcal{P}, \mathcal{C}, \mathcal{G} \rangle$ where F and $\mathcal{S}$ are typed variables that are part of the planning problem. F belongs to the object type that $\mathcal{Q}$ is quantified over, and $\mathcal{S}$ belongs to the object type about which information is to be sensed. $\mathcal{P}$ is a predicate which ensures sensing closure for every pair $\langle f, s \rangle$ such that f is of type F and s is of type $\mathcal{S}$, and both f and s belong to the set of objects in the problem, $\mathcal{O} \in \Pi$; for this reason, we term $\mathcal{P}$ a *closure condition*. $\mathcal{C} = \bigwedge_i c_i$ is a conjunctive first-order formula where each c_i is a statement about the openness of the world with respect to the variable S. For example, `c = (in ?hu - human ?z - zone)` with `S = ?hu - human` means that c will hold for new objects of the type 'human' that are sensed. Finally $\mathcal{G}$ is a quantified goal on $\mathcal{S}$.

Newly discovered objects may enable the achievement of goals, granting the opportunity to pursue reward. For example, detecting a victim in a room will allow the robot to report the location of the victim (where reporting gives reward). Given that reward in our case is for each reported injured person, there exists a quantified goal that must be allowed partial satisfaction. In other words, the universal base, or total grounding of the quantified goal on the real world, may remain unsatisfied while its component terms may be satisfied.

As an example, we present an illustration from our scenario: the robot is directed to "report the location of all injured humans". This goal can be classified as open world, since it references objects that do not exist yet in the planner's object database; and it is quantified, since the robot's objective is to report *all* victims that it can find. In our syntax, this information is encoded as follows:

```
1 (:open
2   (forall ?z - zone
3     (sense ?hu - human
4      (looked_for ?hu ?z)
5       (and (has_property ?hu injured)
6            (in ?hu ?z))
7   (:goal (reported ?hu injured ?z)
8            [100] - soft))))
```

In the example above, line 2 denotes F, the typed variable that the goal is quantified over; line 3 contains the typed variable $\mathcal{S}$—the object to be sensed. Line 4 is the unground predicate $\mathcal{P}$ known as the closure condition (defined earlier). Lines 5 and 6 together describe the formula $\mathcal{C}$ that will hold for all objects of type $\mathcal{S}$ that are sensed. The quantified goal over $\mathcal{S}$ is defined in line 7, and line 8 indicates that it is a soft goal and has an associated reward of 100 units. Of the components that make up an open world quantified goal $\mathcal{Q}$, $\mathcal{P}$ is required[2] and F and S must be non-empty, while the others may be empty. If $\mathcal{G}$ is empty, i.e., there is no new goal to work on, the OWQG $\mathcal{Q}$ can be seen simply as additional knowledge that might help in reasoning about other goals.

Handling OWQGs in the Planning System

To handle open world quantified goals, the planner grounds the problem into the closed-world using a process similar to

[2] If $\mathcal{P}$ were allowed to be empty, the planner could not gain closure over the information it is sensing for, which will result in it directing the robot to re-sense for information that has already been sensed for.

Skolemization. More specifically, we generate *runtime objects* from the sensed variable $\mathcal{S}$ that explicitly represent the potential existence of an object to be sensed. These objects are marked as system generated runtime objects. Given an OWQG $\mathcal{Q} = \langle F, \mathcal{S}, \mathcal{P}, \mathcal{C}, \mathcal{G} \rangle$, one can look at $\mathcal{S}$ as a Skolem function of F, and runtime objects as Skolem entities that substitute for the function. Runtime objects are then added to the problem and ground into the closure condition $\mathcal{P}$, the conjunctive formula $\mathcal{C}$, and the open world quantified goal $\mathcal{G}$. Runtime objects substitute for the existence of S dependent upon the variable F. The facts generated by following this process over $\mathcal{C}$ are included in the set of facts in the problem through the problem update process. The goals generated by $\mathcal{G}$ are similarly added. This process is repeated for every new object that F may instantiate.

We treat $\mathcal{P}$ as an *optimistic closure condition*, meaning a particular state of the world is considered closed once the ground closure condition is true. On every update the ground closure conditions are checked and if true the facts in the corresponding ground values from $\mathcal{C}$ and $\mathcal{G}$ are removed from the problem. By planning over this representation, we provide a plan that is executable given the planning system's current representation of the world until new information can be discovered (via a sensing action returning the closure condition). The idea is that the system is interleaving planning and execution in a manner that moves the robot towards rewarding goals by generating an optimistic view of the true state of the world.

Conclusion

In this paper, we proposed the demonstration of a robotic agent being guided through a complex search and rescue scenario by a planning system. We discussed the motivation behind using a planner in such a task, and described the scenario in detail. We then gave a brief overview of the planning system in use, and the procedure via which the planner's world knowledge is updated. Following this, we visit the problem of updates to the agent's goals, and describe a construct that enables the planner to deal with such updates. We conclude with an explanation of how this construct is accomodated into the planning system.

References

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. *Artificial Intelligence* 173(5-6):562–592.

Cushing, W.; Benton, J.; and Kambhampati, S. 2008. Replanning as a Deliberative Re-selection of Objectives. *Arizona State University CSE Department TR*.

Do, M., and Kambhampati, S. 2002. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proceedings of AIPS*, volume 2.

Etzioni, O.; Golden, K.; and Weld, D. S. 1997. Sound and efficient closed-world reasoning for planning. *AIJ* 89(1-2):113–148.

Talamadupula, K.; Benton, J.; Schermerhorn, P.; Scheutz, M.; and Kambhampati, S. 2010. Integrating a Closed-World Planner with an Open-World Robot. In *AAAI 2010*.

Job Scheduling in Lean Document Production

Rong Zhou[1], **Sudhendu Rai**[2], **Minh Do**[1]

[1]Palo Alto Research Center
Palo Alto, CA 94304, USA
{rong.zhou,minh.do}@parc.com

[2]Xerox Research Center Webster
Webster, New York 14580
sudhendu.rai@xerox.com

Abstract

Lean Document Production (LDP) is a novel class of productivity-enhancement offerings that were originally invented at the Xerox Research Center Webster (XRCW) for the $100 billion printing industry in the United States. Implemented by Xerox in over 100 sites to date, LDP has provided dramatic productivity and cost improvements for both print shops and document-manufacturing facilities, as measured by reductions of 20∼40% in revenue-per-unit labor cost. LDP has generated over $200 million of incremental profit across the Xerox customer value chain since its initial introduction in 2000. In the past three years, PARC's Embedded Reasoning Area has been collaborating with XRCW to extend the scheduling capabilities of the LDP toolkit. We describe a number of newly added features such as adaptive batch splitting, multi-site scheduling and multi-core parallelization that have significantly improved the performance of our AI search-based scheduler for print shops of all sizes, particularly for those large document-production facilities that can process thousands of monthly jobs on a diverse set of document-production equipment with non-uniform speed and sequence-dependent setup times.

Introduction

The provision of services that improve business productivity is a major component of Xerox's growth strategy. These services include the outsourcing and improvement of customer print-shop operations. Since 1999, Xerox has invented, tested, and implemented a novel class of productivity-improvement offerings, trademarked LDP Lean Document Production® solutions (Rai *et al.* 2009), for the printing industry. The size of the market for these offerings, which have created dramatic productivity and cost improvements for both print shops and document-manufacturing facilities, is $100 billion in the United States alone. They have greatly expanded the applications of automated scheduling tools and operations-research techniques in the printing industry.

Xerox Corporation participates in the printing industry in a number of ways; one is as a provider of services, via Xerox Managed Services (XMS), to manage print operations for clients who choose to outsource their in-plant print operations, called *in-plants*. In the 1990s, Xerox was extremely successful in growing revenues and profits in this segment by utilizing highly automated printing and reprographics equipment as a vehicle to offer print-shop outsourcing at a much lower cost than was characteristic of the typical in-plant of the day. However, other firms have since offered comparable, highly automated equipment and outsourcing services based on the use of such equipment. Thus, XMS revenues and margins came under significant pressure. LDP was conceived and invented as a set of offerings that would reestablish Xerox's reputation as the leading print-shop productivity enhancer in the industry and, in the process, increase XMS revenues and profits.

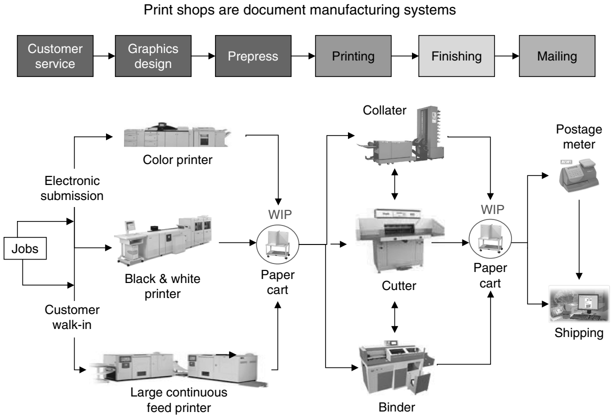


Figure 1: Print-shop workflow (WIP stands for work-in-progress).

Workflow in Print-Shop Environment: Print shops can be classified into three categories based on the activity that they perform: transaction printing, on-demand publishing, or a combination of both. LDP solutions encompass all three print-shop domains. Typically, each of the six steps in the print production workflow is associated with a specific department: (1) customer service and production planning, (2) graphics design, (3) prepress, (4) printing, (5) finishing department, and (6) mailing. Any design and operations methodology for print production must comprehend both digital and off-set printing workflows independently and when they coexist. Figure 1 shows the various operations that are performed in typical print-shop workflows.

Challenges in Print-Shop Productivity Improvement

Document production has unique characteristics that make it difficult to operate print shops efficiently.

- *Long bid times*: Customers often want to see physical proofs before committing to the entire print job.
- *Variability in workflow types*: The workflow required to process print jobs varies considerably from job to job.
- *Variability in job-size distribution*: Print shops experience significant fluctuations in demand and jobs submitted to even the same shop can have sizes (e.g., number of pages) that differ by several orders of magnitude.
- *Variability in equipment capability and speed*: Print shops can host a large number of non-uniform devices, each of which may have some unique capabilities and run at various speeds under different operation modes or conditions.
- *Variability in setup times*: The setup time of even a single document-production device can change dramatically, depending on the kind of job that was processed previously on the device.
- *Variability in labor and equipment*: Print shops are often labor intensive, with many manual processing steps.
- *Departmental production and scheduling*: Print shops typically organize their equipment and labor into specific functional departments to improve utilization of resources and maintain a labor force skilled in specific tasks.

These characteristics pose significant challenges for developing a standardized productivity-improvement methodology that is scalable and adaptable across multiple printing environments.

LDP Scheduling

In this section, we describe how LDP selects and schedules jobs for the print shop. Figure 2 shows LDP's two-level architecture in which jobs submitted to the shop are distributed to one or more "mini-shops" called *cells* for production (Rai and Viassolo 2003). This is a radical departure from traditional department-style print shop designs in which equipment performing the same or similar functions is clustered together (e.g., all the printers reside in a "printers-only" area, all the finishing devices in a "finishers-only" area, and so on). In contrast, a cell in LDP normally contains a diverse set of machines that can perform different functions.

LDP's cellular design methodology effectively unites various machines and human operators involved in different stages of document production to significantly boost print-shop productivity. The reason is that a cellular design allows each cell to be optimized separately for a better match of the characteristics of the print jobs to the production capabilities of the cell. As a beneficial side effect, work-in-progress (WIP) in LDP-enhanced shops is usually much lower than traditional shops, thanks to the close proximity of the machines residing in the same cell.

Figure 3 shows the block diagram of LDP's scheduling system, which takes as inputs a shop definition file and a jobs definition file.

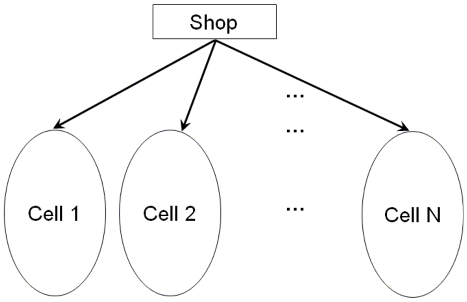


Figure 2: LDP's two-level architecture.

Shop definition: The shop definition file contains information regarding all aspects of the shop, which most closely resembles the domain (i.e., operators) file in PDDL planning. More specifically, a shop comprises of

- *Schedule*: This is the shop-level schedule that describes the operational hours of the entire shop (e.g., 8am ∼ 5pm everyday, except for weekends), which can be used as the default schedule for the machines and human operators in the shop.
- *Sequencing policy*: The sequencing policy determines the order in which the jobs are scheduled. Currently, the system supports the following policies: (1) first in first out, (2) earliest due, (3) least slack, and (4) minimum processing time. Of course, more policies can be added, which can be done easily with the current implementation.
- *Machines*: Each machine is identified with a unique name, and is capable of performing a set of function sequences with various speeds, setup times (which may depend on the attributes of the previous job), and pricing information. Optionally, each machine can have its own schedule (e.g., scheduled maintenance between 3∼4pm on Thursday), which overrides the default shop-level schedule.
- *Operators*: Each human operator is identified with a unique name and possesses a set of skills for performing manual steps (e.g., inspection) and supervising machine operations, which are identified by the names of the machine function sequences the operator knows how to operate (e.g., color printing on a continuous-feed printer). In addition, each operator can have his or her own schedule (e.g., working from 9am to 4pm Monday through Friday) that overrides the default shop-level schedule.
- *Cells*: Each cell is made up of a group of human operators and a list of machines they operate or supervise. Individual cells can enable or disable batch splitting, which allows the division of a big job into smaller pieces called *batches*, to further improve throughput. For scheduling flexibility, each cell can choose to ignore the schedules of its operators when assigning jobs to machines, assuming the production schedule is machine-bound instead of operator-bound. The default scheduling option, however, requires that the schedules of both the machines and the human operators be taken into account.

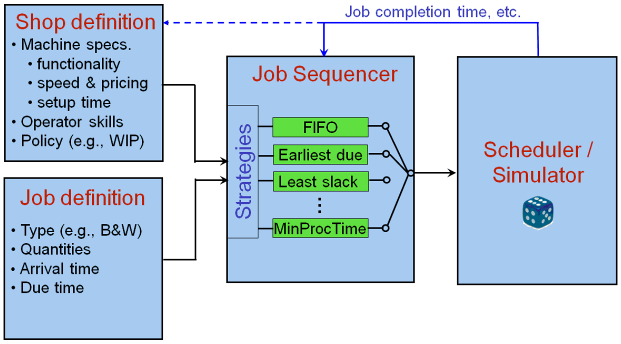


Figure 3: LDP scheduling system overview.

Jobs definition: The jobs definition file contains detailed information regarding the list of jobs submitted to the shop, which most closely resembles the problem instance (i.e., facts) file in PDDL planning. More specifically, a job comprises of

- *Temporal constraints*: The arrival and due dates of a job are provided when the job is submitted.
- *Resources*: Each resource is identified by a unique name (within the job) and describes the quantities required to complete a particular document-production step (e.g., 100 color pages must be printed for the color-print step)
- *Function steps*: Each function step is a unit operation (e.g., color printing on letter-sized paper), which has a (possibly empty) set of input resources and produces a (possibly empty) set of output resources. This is similar to the precondition and effects in STRIPS, except that the duration of each step is non-uniform, depending on the quantities defined in the resources. The steps themselves do not have to form a linear sequence of actions, as in sequential planning. Parallel steps are quite common in print-shop operations (e.g,. the front, the body, and the back matter of a book can be produced simultaneously). Each step can have a set of attributes, which are used to compute the setup time as follows: for each attribute that is changed (compared to the attribute of the last job that was processed on the same machine), a corresponding penalty is added to the setup time.

To schedule multiple jobs, LDP first invokes a job sequencer that orders these jobs according to one of the four sequencing policies shown in the middle of Figure 3. Jobs appearing earlier in the sequence are scheduled before those that appear later. Once the sequence of jobs has been decided, they are fed to a scheduler one at a time such that each subsequently scheduled job must respect all the constraints imposed by those jobs that were scheduled before it. This is obviously a greedy policy that does not guarantee global optimality. However, in practice we have found it works reasonably well in various print shop configurations and job mixes. When faced with multiple scheduling choices, the scheduler always picks the one that finishes the last job as early as possible, essentially favoring plans with a shorter makespan. Note that a similar greedy-search strategy was also used in the Tightly Integrated Parallel Printing (TIPP) project that the PARC researchers have worked on previously (Do *et al.* 2008; Ruml *et al.* 2011).

Upon the completion of the last job in the sequence, the scheduler returns a number of statistics designed to summarize the quality of the schedules found. These statistics include average processing and job turnaround times, average and maximum lateness, and the number of late jobs, among others. They can be used as part of a feedback loop (as indicated by the dashed line in Figure 3) to improve the layout of a shop, because as the job mix changes, so must a cellular design for the shop. In an earlier implementation, the toolkit uses a stochastic simulator to generate the performance statistics, which can vary slightly from one run to another. The search-based scheduler developed by us is the first deterministic scheduler for LDP. From a practical implementation viewpoint, having a deterministic scheduler makes it easy to operationalize our toolkit, since the resulting schedules are free of any idiosyncracy produced by a particular run of the scheduler.

Advanced Features in LDP Scheduling

We next describe a number of advanced features that are first made available in our deterministic scheduler.

- *Adaptive batch splitting*: An important throughput-enhancement strategy in LDP is batch splitting, which chops a large job into a number of smaller units called "batches." The idea is to eliminate downstream waiting as soon as some portion of a long job is ready for further processing. In an earlier version, the batch size is calculated using fixed formulae that do not adapt to the dynamic workload of each cell. Later on, we designed a fully adaptive strategy that first sub-divides a long job into sufficiently many batches, followed by a merge phase in which the algorithm recursively combines two batches that can be scheduled back to back on the same machine, to ensure only a minimal number of batches are created.
- *Multi-site scheduling*: We extended our basic single-site scheduler to a distributed production environment in which multiple geographically separated sites can efficiently coordinate with one another to share the workload while respecting all their individual resource and sequencing policy constraints. It takes into account the transportation delays between multiple sites when making scheduling choices. Our computational results show huge reductions in the number of late jobs and average turn around time compared to the single-site equivalent that treats each site as an isolated shop. Multi-site scheduling enables better resource utilization and cost reduction, currently a popular trend in the printing industry.
- *Multi-core parallelization*: To support efficient parallelization of the scheduler on modern processors, We developed a multi-core version of our scheduler that uses shared-memory parallelization (based on POSIX threads) to achieve near linear speedup in the number of processor cores used. This is particularly beneficial for multi-site

scheduling, which typically deals with thousands of jobs and a number of cells in each site. Compared to the earlier versions (implemented in Java), our latest C++ implementation is not only much faster as measured in wall-clock times, it is also significantly more memory-efficient. This allows our scheduler to handle larger shops with many more jobs than its predecessors.

Lessons Learned

There are a number of valuable lessons that we learned in the process of developing the LDP scheduler. We highlight a few below.

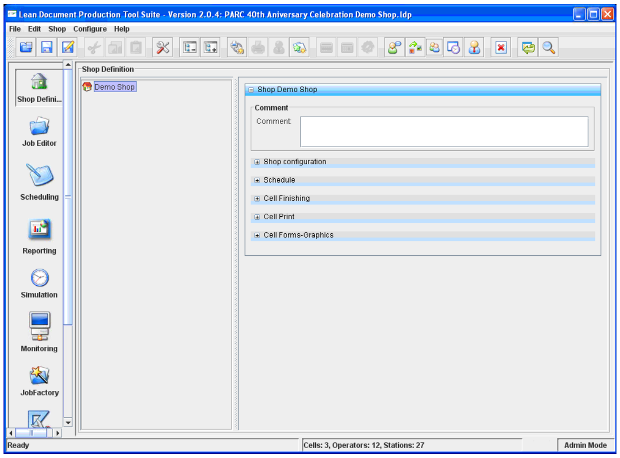
First, we found that real-world data of print shops and jobs typically contain a great deal of noises such as inconsistent or missing fields. As a result, we had to spend a lot of time on data cleaning and consistency checking to make sure they accurately model the real print-shop environments and jobs. To mitigate such a laborious and error-prone task, we developed an automated consistency checking tool and embedded it inside our scheduler. The tool has been invaluable to us, as it uncovered a number of modeling bugs in existing LDP scenarios that were supposed to be already cleaned.

Second, a well-designed GUI can be crucial to customer adoption and can have a significant impact on the overall productivity of the system. With an earlier version of the toolkit, the feedback we received from the customers was that it could take a while to use all the LDP functionalities and the learning curve was somewhat steep. In the later releases, efforts have been made to simplify and streamline the GUI to make it more accessible to average users. This is very well received by the customers. Figure 4 shows a sample screenshot of the LDP toolkit in its shop definition mode. As shown in the left panel, the toolkit includes the *Job Editor*, *Scheduling*, *Reporting*, *Simulation*, *Monitoring*, *Job Factory*, as well as other tools.

Third, we found it interesting that although the modeling language used by LDP does not resemble much of a "domain-independent" planning language used in the research community, it is actually quite adequate for the print-shop scheduling applications. More surprisingly, it appears that this somewhat domain-specific language can be extended (at a reasonable cost) to other scheduling applications beyond printing, such as the generic job-shop scheduling problems (Pinedo and Chao 1998) found in many other industries. This experience has taught us a lesson on where to strike a good balance between domain independence and scheduler efficiency. From the end user's perspective, which planning language to use is likely a low-visibility issue, since all they interact with is the GUI, and whether it is PDDL or LDP's XML-based, domain-specific language under the hood is of little concern to the end user.

Conclusion & Future Work

We have presented a real-world print shop productivity enhancement tool called LDP that has generated significant revenues for Xerox and its customers. It has roots in cellular manufacturing seen in the automobile industry yet its solutions have all been successfully adapted for and validated by the printing industry.



Lean Document Production Tool Suite - Version 2.0.4: PARC 40th Aniversary Celebration Demo Shop.ldp

Ready

Cells: 3, Operators: 12, Stations: 27

Figure 4: LDP toolkit screenshot.

In the future, we plan to take the same approach and practice in lean manufacturing, as embodied by our LDP toolkit, to other application domains with similar characteristics. We believe our experience in bringing simple yet effective scheduling techniques to realistic production systems has values beyond the document-production world.

References

Minh B. Do, Wheeler Ruml, and Rong Zhou. Planning for modular printers: Beyond productivity. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS-08)*, pages 68–75, 2008.

Michael Pinedo and Xiuli Chao. *Operations Scheduling with Applications in Manufacturing & Services*. McGraw-Hill, Boston, 1998.

Sudhendu Rai and Daniel E. Viassolo. A lean document production controller for printshop management. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 4, pages 4115–4125, 2003.

Sudhendu Rai, Charles B. Duke, Vaughn Lowe, Cyndi Quan-Trotter, and Thomas Scheermesser. LDP lean document production – O.R.-enhanced productivity improvements for the printing industry. *Interfaces*, 39(1):69–90, 2009.

Wheeler Ruml, Minh B. Do, Rong Zhou, and Markus P.J. Fromherz. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research*, 40:415–468, 2011.