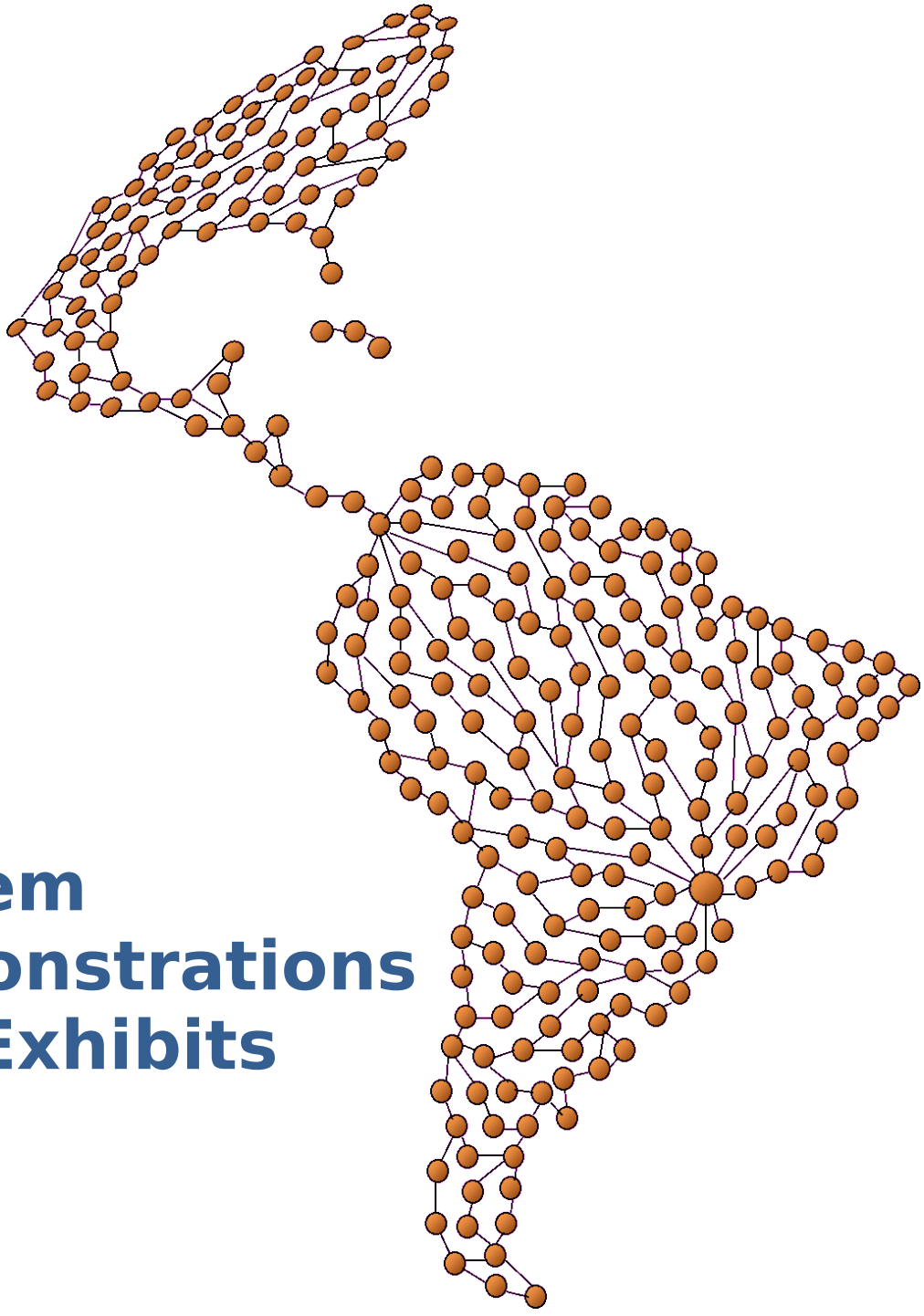
22nd International Conference on Automated Planning and Scheduling

June 25-29, 2012, Atibaia – Sao Paulo – Brazil



System Demonstrations and Exhibits

Edited by
Simone Fratini and Adi Botea

Organization

Simone Fratini, ESA-ESOC, Germany
contact email: simone.fratini@esa.int
Adi Botea, IBM Research, Dublin, Ireland
contact email: adibotea@ie.ibm.com

Foreword

System demonstrations have a marked role in the series of annual ICAPS conferences. The event is one of the incentives that ICAPS offers towards proving the practical usefulness of planning and scheduling research. In 2012, the System Demonstrations and Exhibits event features 9 demonstrations. For certain types of submissions, providing an extended abstract was optional. This includes work also presented in the main ICAPS track, and entries that also participate in the 4th International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS). As a consequence, the current set of proceedings contains 6 extended abstracts and 3 shorter abstracts. We thank the authors for the impressive work behind their on-site demonstrations.

Simone Fratini, Adi Botea
June 2012

Contents

DUKC Optimiser: Maximising Cargo Throughput at a Bulk Export Port

Elena Kelareva
OMC International / ANU / NICTA
Melbourne, Australia
elena.kelareva@nicta.com.au

Abstract

This demo presents DUKC® Optimiser – a system for maximising cargo throughput at a bulk export port by scheduling sailing times and drafts for a set of ships. An earlier prototype of the system underwent user testing in 2010 (Kelareva 2011), and a number of improvements resulting from user feedback have been incorporated in this updated version.

DUKC® Optimiser is the first system for automatically scheduling ship sailing times and drafts at a bulk export port which takes into account time-varying draft restrictions that take into account live environmental conditions. The system uses the Dynamic Under-Keel Clearance (DUKC®) software developed by OMC International to calculate draft restrictions. These restrictions are then converted to a contraint programming model, and solved using the G12 finite domain solver, developed by NICTA.

The software is able to find optimal schedules for realistic problem sizes, and is able to produce schedules which allow ships to carry more cargo than would be permitted by traditional constant-draft or manual scheduling approaches.

1 Introduction

At a bulk export port, the port authority aims to maximise cargo throughput at the port while maintaining safety. One key aspect of safety is restrictions on ship draft. Draft is the distance between the waterline and the bottom of the ship's keel, which increases as more cargo is loaded. Most ports have restrictions on maximum draft for ships entering and leaving the port, as loading a ship beyond the safe draft limit may result in the ship running aground.

At most ports, draft restrictions vary over time, and depend on the estimated under-keel clearance (UKC – amount of water under the keel) which varies with tide, wave, current and wind conditions. The Dynamic Under-Keel Clearance (DUKC®) software developed by OMC International has been very effective at increasing both maximum draft and safety by improving accuracy of UKC modelling at ports, thus reducing the conservatism required to maintain safety (OMC International 2009). Many ports worldwide now use DUKC® software to calculate draft restrictions, as this enables more cargo to be loaded onto ships without compromising safety.

Scheduling ship sailing times and drafts at bulk export ports is currently done manually or using simple tools such as Microsoft Excel. When ports use static under-keel clearance constraints that depend only on long-term tide forecasts, these constraints are same for all ships and don't need to be adjusted as enviro data is updated. However, with DUKC® software, the draft constraints vary between ships, and may be updated as enviro predictions change. DUKC® predicts under-keel clearance more accurately, thus enabling more cargo to be loaded onto ships, but it makes scheduling more complex and may require changes to the schedule if conditions change.

DUKC® Optimiser is a new tool that aims to simplify scheduling of ship sailing times and drafts for bulk export ports that use DUKC® software to calculate draft restrictions. DUKC® Optimiser also aims to find optimal schedules that allow ships to carry more cargo than schedules produced by human schedulers.

2 DUKC® Optimiser Background

A command-line prototype of DUKC® Optimiser was developed and tested by port schedulers in late 2010, and demonstrated at ICAPS 2011 (Kelareva 2011). An updated model containing improvements based on user feedback was incorporated in a commercial system in 2012 (Kelareva et al. 2012a).

The major improvement to the model in this version was the introduction of constraints on the availability of tugs – small boats that are used to assist ships to enter or leave port. User testing in late 2010 found that tug availability could constrain the schedule, so schedules produced by the model without tug constraints could be infeasible in practice. Tug constraints therefore needed to be incorporated before the system could be used in practice.

Another major improvement was improving the speed of the model, as described in (Kelareva et al. 2012a) and (Kelareva et al. 2012b).

The initial prototype was command-line based, which was sufficient to gather initial user feedback on schedule quality, but would have been inconvenient for operational use. The scheduling system was therefore incorporated into a commercial web-based dynamic under-keel clearance management system - DUKC® Series 5, developed by OMC International.

Vessel Beam, LBP	Sailing Draft	Sailing Slot Open Time	Sailing Slot Close Time
(A) AMIRA (32.26m, 217m)	16.00m	25May2012 0910	25May2012 0925
(B) BEGONIA (45m, 280.8m)	18.00m	25May2012 1235	25May2012 1250
(C) CAMELLIA (50m, 300m)	**18.75m**	25May2012 1145	25May2012 1200
(D) DIONE (45m, 283m)	18.00m	25May2012 1020	25May2012 1035
(E) EURYDICE D (42.5m, 267.6m)	18.00m	25May2012 1050	25May2012 1105
(F) FIRST JUPITER (45m, 277m)	**17.45m**	25May2012 1255	25May2012 1310

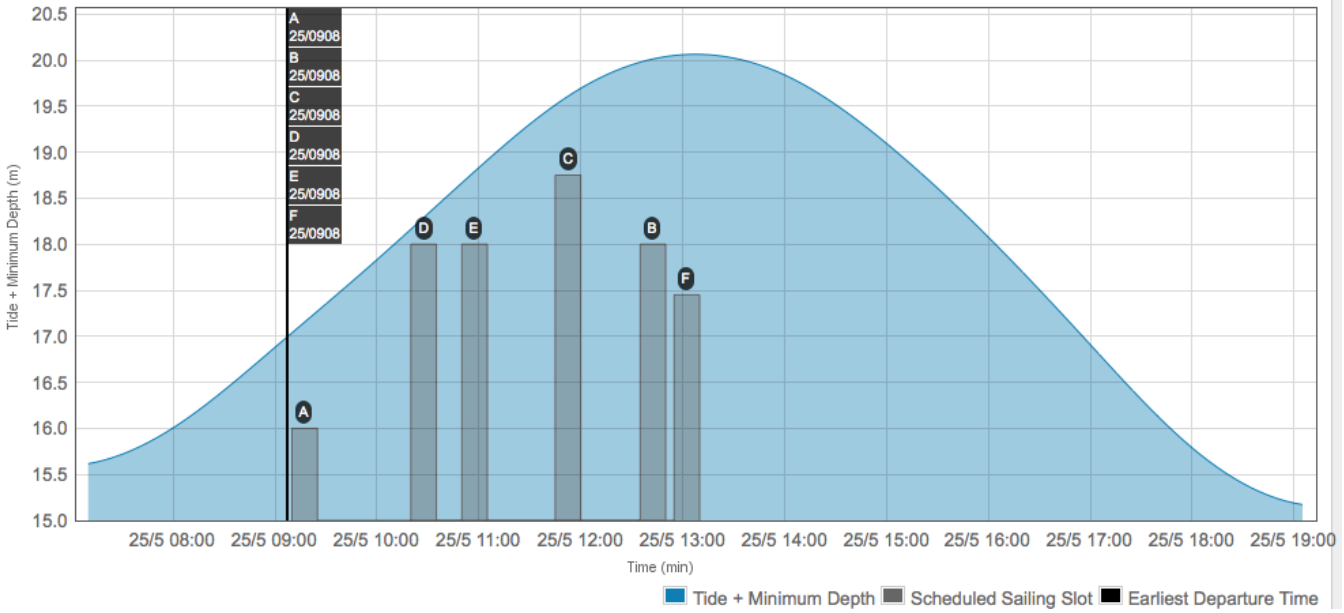


Figure 1: DUKC® Optimiser Result

3 Implementation

3.1 User Interface

To run a DUKC® Optimiser calculation, the scheduler must enter parameters such as length and beam for each ship that are used to calculate under-keel clearance. Other inputs required to calculate a schedule include the earliest sailing time for each ship, the number of tugs required for each ship, the range of drafts to calculate, and a priority number that is used to ensure fairness to different companies using the port.

Figure 1 shows an output schedule for a set of six ships sailing on one tide. In the graph on the bottom half of the screen, each bar represents a ship, with the height of the bar corresponding to the draft that the ship is scheduled to sail with, and the location of the bar along the x-axis indicating the time at which the ship is scheduled to sail. Each ship is scheduled with a time slot of 15 minutes, rather than a fixed time point, as it is impractical to expect a large bulk carrier to sail precisely at a given minute.

The blue curve indicates the minimum water depth along the channel plus the height of the astronomical tide prediction at that point in time. This does not take into account wave response, squat, heel, safety factor, or variation from the astronomical tide prediction, all of which can decrease the amount of water available to the ship. This results in the height of the blue curve being significantly above the height of the bars indicating ship draft.

3.2 Other Features

The web-based DUKC® Series 5 system includes a number of additional features to assist port schedulers, pilots and harbourmasters in making ship scheduling and sailing decisions. Port administrators may limit user permissions to only access schedules for ships belonging to their organisation, or to only access features that are required for their job. The system can also display live environmental data, such as measurements from wave buoys and tide gauges. The system also includes the DUKC® calculations used to provide under-keel clearance advice to pilots and ship captains immediately prior to sailing, and an in-transit monitoring tool that tracks the locations of ships at the port.

3.3 System Architecture

When the scheduler selects a set of ships to be scheduled in the web-based GUI, a query is sent to the DUKC® Optimiser server. Upon receiving a schedule query, DUKC® Optimiser converts it into a set of queries to OMC's DUKC® software. The DUKC® software uses real-time environmental forecasts and measurements to analyse each ship's motion, and thus to calculate the ship's under-keel clearance – the amount of water under the ship at each point in the transit. This produces 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 Constraint Programming (CP) model, implemented in the MiniZinc opti-

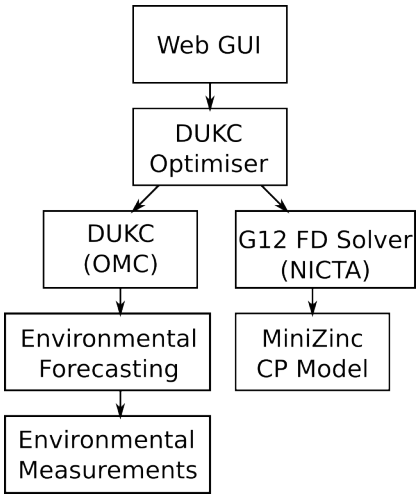


Figure 2: DUKC® Optimiser System Architecture

misation language (Nethercote et al. 2007). This model is then solved using the G12 finite domain solver (Stuckey et al. 2005). The GUI then displays the resulting schedule.

3.4 Constraint Programming Model

The constraint programming model, including speed improvements from the older command-line version, is discussed in detail in (Kelareva et al. 2012a). The model is only described briefly here.

Basic Model The decision variables in the Constraint Programming model used to create schedules are the sailing times for each ship. The maximum draft for each ship is a function of time, specified at 5-minute intervals, as the maximum draft allowed by the DUKC® may change rapidly.

The main constraints of the original model without tugs are:

- Constraints on the earliest time when each ship may sail.
- Constraints on the availability of berths for incoming ships.
- Constraints enforcing minimum separation time between successive ships.

Tug Constraints Tug constraints proved to be very difficult to implement efficiently, as tug job durations depend on both the ship the tug is working on, and the ship it will work on next. After several unsuccessful attempts, we found an implementation that was able to solve realistic problem sizes within 5 minutes by splitting the sequence of ships into four types of scenarios, as shown in Figure 3, and calculating tug constraints separately for each scenario.

Objective Function The objective function may vary between ports – some ports may only maximise throughput; other ports may prioritise fairness to competing clients above maximising total throughput for the port. A port objective function may also need to take into account shipping contracts used by shippers at the port, as these may affect the cost and benefit to shippers of sailing with more or less draft.

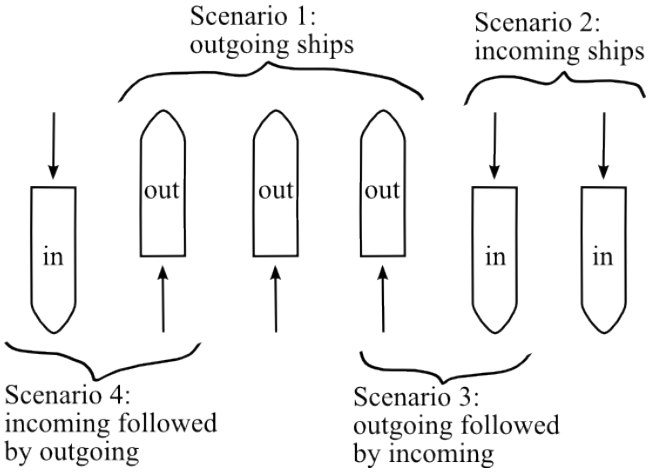


Figure 3: Scenarios for Tug Constraints

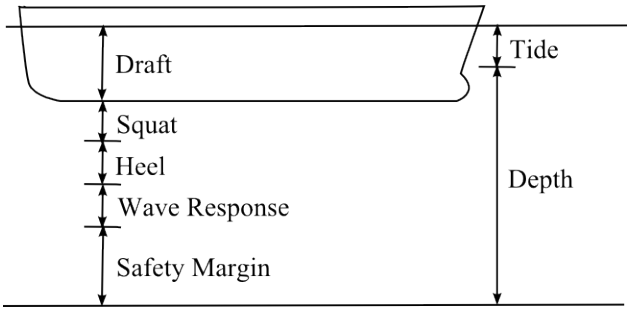


Figure 4: Dynamic Under-Keel Clearance Components

3.5 Dynamic Under-Keel Clearance

Figure 4 illustrates components of ship motion taken into account by the DUKC® software. These include:

- **Draft:** the distance from the waterline to the bottom of the ship's keel.
- **Squat:** a phenomenon 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:** motion resulting from the action of waves on the ship. Only the vertical component of this motion affects under-keel clearance.

Under-keel clearance is computed as follows:

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

If the under-keel clearance is below the required safety limit, then the DUKC® software will advise the operator not to sail. However, 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).

4 Benefits

Existing ship scheduling approaches either leave draft constraints entirely up to human schedulers (Fagerholt 2004),

or use simple constant draft constraints that do not vary with time (Christiansen et al. 2011) (Song and Furman 2010). Scheduling of ship sailing times at a port is usually done manually by human schedulers following simple heuristic rules such as scheduling the ship with the largest maximum draft first, and scheduling each ship at the earliest time it can sail (Kelareva et al. 2012a).

Both of these approaches can lead to suboptimal schedules where ships carry less cargo than the maximum. An example presented by (Kelareva et al. 2012a) shows that even for a simple schedule with three ships, fixed-draft and manual scheduling approaches can fail to find the optimal schedule, resulting in 10cm less total draft. An average Capesize iron ore carrier can transport 130 tonnes of iron ore per centimetre of draft (Port Hedland Port Authority 2011), so this results in around US$221,000 less iron ore being transported on the three ships, at the January – October 2011 average iron ore price of around US$170/tonne (Index Mundi 2011).

This small example clearly shows the financial benefit of using accurate time-varying draft constraints and optimal schedules. These are simple examples with only three ships, all berths at the same location along the transit, and no tug constraints taken into account. Real schedules would be even more complex and difficult to optimise manually.

5 Conclusions

We have presented DUKC® Optimiser – a web-based system for maximising throughput at a bulk export port by scheduling ship sailing times and drafts. It uses the Dynamic Under-Keel Clearance (DUKC®) software developed by OMC International to calculate constraints on allowable drafts for each ship at each point in time, taking into account the effects of tide, waves and current on ship motion.

DUKC® Optimiser contains a constraint programming (CP) model implemented in the MiniZinc optimisation programming language, which is solved using the G12 finite domain solver developed by NICTA. Major constraints include sequence-dependent separation times between ships and constraints on the availability of tugs.

The system has undergone user testing in 2010 (Kelareva 2011), and user feedback has been incorporated into an updated version (Kelareva et al. 2012a).

A comparison of optimal schedules produced by DUKC® Optimiser against constant-draft ship scheduling approaches and schedules produced by simple heuristics used in practice at ports has demonstrated that DUKC® Optimiser is able to find optimal schedules which allow ships to load more cargo than either fixed-draft or naive manual scheduling approaches (Kelareva et al. 2012a). This shows that DUKC® Optimiser may provide a large benefit to industry, as every centimetre of extra draft allows more cargo to be carried on the same set of ships, thus reducing transportation costs.

6 Acknowledgements

DUKC® Optimiser is developed by OMC International. The author would like to acknowledge the contribution of other OMC engineers involved in the development of DUKC® Optimiser, particularly Gordon Lindsay, 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.; Flatberg, T.; Haugen, Ø.; Kloster, O.; and Lunda, E. 2011. Maritime inventory routing with multiple products: A case study from the cement industry. *European Journal of Operational Research* 208(1):86–94.

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

Index Mundi. 2011. Iron ore monthly price. http://www.indexmundi.com/commodities/?commodity=iron-ore.

Kelareva, E.; Brand, S.; Kilby, P.; Thiébaux, S.; and Wallace, M. 2012a. CP and MIP methods for ship scheduling with time-varying draft. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'12)*.

Kelareva, E.; Kilby, P.; Thiébaux, S.; and Wallace, M. 2012b. Ship scheduling with time-varying draft. In *Fifth International Workshop on Freight Transportation and Logistics (ODYSSEUS'12)*.

Kelareva, E. 2011. The "DUKC Optimiser" ship scheduling system. In *2011 International Conference on Automated Planning and Scheduling: System Demonstration*.

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.

OMC International. 2009. DUKC helps Port Hedland set ship loading record. http://www.omc-international.com/images/stories/press/omc-20090810-news-in-wa.pdf.

Port Hedland Port Authority. 2011. Dynamic under keel clearance system. http://www.phpa.com.au/dukc_information.asp.

Song, J.-H., and Furman, K. 2010. A maritime inventory routing problem: Practical approach. *Computers & Operations Research*.

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.

Planning-Based Composition of Stream Processing Applications

Mark D. Feblowitz, Anand Ranganathan, Anton V. Riabov, and **Octavian Udrea**
IBM T. J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA

Abstract

In this demonstration we present our planning-based tools for software composition, and in particular, for composition of distributed stream processing applications. The applications composed by our tools are deployed on IBM InfoSphere Streams distributed stream processing middleware. The applications are used to process large data volumes in real time on large clusters of commodity servers. Our tools include MARIO, the goal-driven planning-based application composition tool for business users, and an Eclipse-based integrated development environment (IDE) for developing planning domain descriptions.

Overview

Distributed stream processing middleware, such as IBM InfoSphere Streams (IBM InfoSphere Streams), allows software developers to apply the full computational power of numerous commodity servers toward large-scale real-time analysis of streaming data. Recently, applications that place high demands on the rate and volume of input data have emerged in telecommunications, finance, health care, and other industries. Stream processing applications are developed by specifying, in a programming language, the data flows between inputs and outputs of stream processing operators.

To significantly shorten the development cycle and make applications more flexible and responsive to changing business needs, we have developed a planning-based approach that gives business users the ability to assemble stream processing applications for their needs on the fly, without programming or using drag-and-drop interfaces. Similarly to planning-based approaches that were proposed for Web Service composition, e.g. (Traverso and Pistore 2004), we have extended AI planning formulations and techniques to address the problem of composing stream processing applications.

To further simplify the goal-driven composition process, we have developed an iterative goal refinement approach that gives business users the ability to express their goals and explore alternative compositions by choosing from a set of business-relevant terms. The built-in optimization engine helps assemble the best match even for ambiguous goals given by the end users. Developers, on the other hand, can easily describe and publish new operators and data sources for use in new compositions. Our tools, including the web-based goal-driven application composition tool (MARIO), and the planning domain development environment (IDE), have been used in a customer pilot since 2009.

MARIO: A Goal-Driven Application Composition Tool

We have developed MARIO (Bouillet et al. 2008) as a framework for automated composition of analysis flows, with the primary motivation to compose stream processing applications. We have further generalized MARIO to compose and deploy analysis flows on a variety of other platforms, including Web Services, Enterprise Service Bus, and others. MARIO can be extended to support new platforms by adding new plug-ins that generate code and deploy composed flows.

An overview of MARIO and its interactions with business users and other systems is shown graphically in Figure 1. MARIO interacts with the business users via a Web-based interface to receive and refine goals, generates and deploys application code for user-specified goals, and presents the results of execution back to the user.

In MARIO, composition goals are specified as sets of business-relevant keywords (i.e., tags). For example, a financial analyst may request an application that identifies stocks sold below volume-weighted average price (VWAP), and therefore constituting a bargain, by selecting tags "*VWAP, BargainIndex*" as a goal.

A unique feature of MARIO is interactive goal refinement. Possible refinements of the specified goal are generated by the MARIO planner together with the optimal plan, based on the analysis of alternative plans. For example, the planner may suggest *LinearIndex* and *ExponentialIndex* as possible refinements of "*VWAP, BargainIndex*", if there are alternative plans matching to those tags. The users are allowed to specify ambigous goals, allowing the planner to make remaining decisions based on its optimality criteria. On the other hand, the users can also refine goals until there are no more refinements.

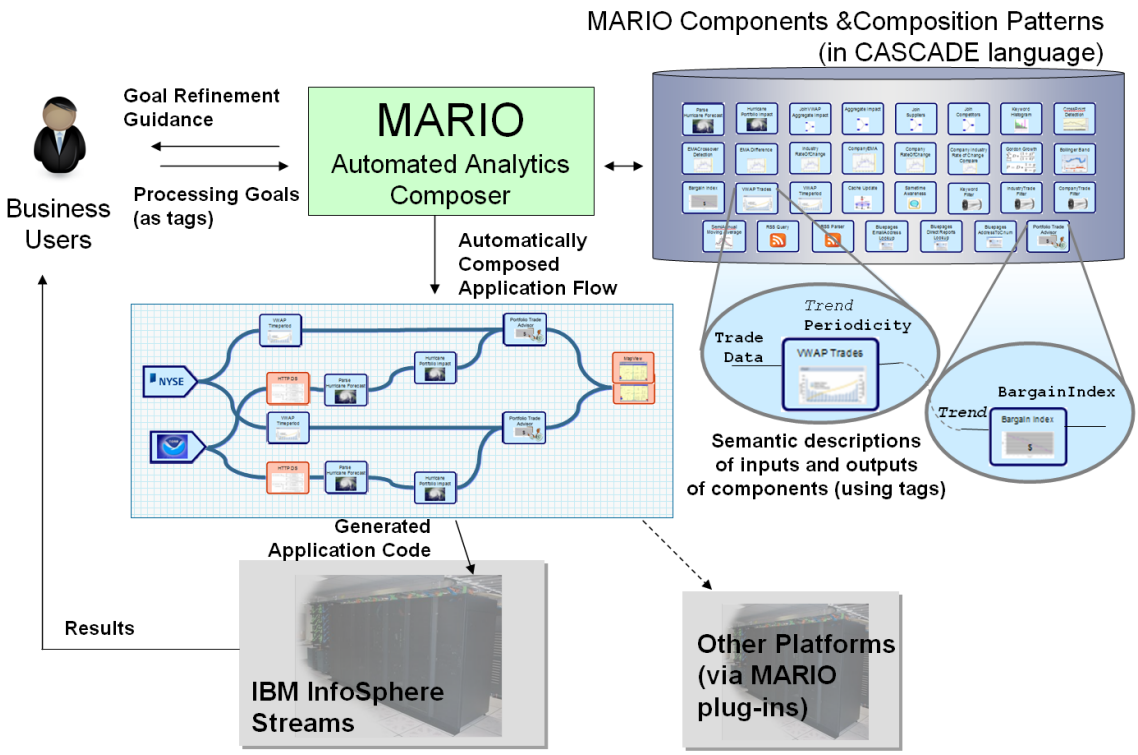


Figure 1: MARIO Overview

In general, goal tags match tags used by the developers in describing inputs and outputs of stream processing operators and data sources. The developers can define sub-tag relationships between tags, allowing MARIO to use reasoning during planning, and match specific annotation tags to general goal tags. For example, tags *LinearIndex* and *ExponentialIndex* can be declared to be sub-tags of *BargainIndex*, and, as a result, the plans that match either tag will also match any goal that includes *BargainIndex*.

In addition to composing applications, which we consider its primary functionality, MARIO Web server includes supporting functions. For business users, it is the primary console for managing and deploying stream processing applications, providing the user interface and server-side functionality for managing long-running applications in a multi-user environment.

Planning Approach

To respond to composition requests efficiently, MARIO relies on a fast special-purpose optimizing planner that solves planning problems described in SPPL domain description language (Riabov and Liu 2006). SPPL descriptions are automatically generated from user-specified goals and developer-defined application composition domains.

In MARIO, tags are used for describing the semantics of analytics and data sources, as well as for specifying the goals. In the past we have implemented a more general OWL-based model (Liu, Ranganathan, and Riabov 2007). However, the OWL-based model was difficult to use in practice, since few stream processing application developers were familiar with OWL. Our current tag-based model is designed to be easier to understand and use, while preserving a few basic reasoning capabilities, such as the sub-tag relationship.

The developers use the Cascade language (Ranganathan, Riabov, and Udrea 2009) to describe application composition domains. For example, the output of a stream processing operator computing exponential index can be tagged with *ExponentialIndex*, and the operator itself can be placed, as one of the possible choices, within a Cascade composition pattern flow graph.

The goals specified by the business users as tags are automatically translated to SPPL problem descriptions. Since goal refinement is the only method for specifying goals, only valid goals can be specified, and only known tags can be included in goals, making this translation simple. Cascade composition patterns and tag relationships are also compiled into SPPL domain descriptions for planning.

MARIO finds an optimal plan and analyzes alternatives every time a user adds or removes a goal tag. This allows users to explore the space of possible plans. However, to make goal refinement process truly interactive, and to serve multiple users simultaneously, planning must be efficient. Our SPPL planner (Riabov and Liu 2006) implements multiple performance optimizations, and achieves planning times of a few seconds for most practical applications.

Goal Tags	Planning Time (seconds)
$\emptyset$	0.11
VWAP	0.16
VWAP, BargainIndex	0.09
VWAP, BargainIndex, ExponentialIndex	0.10
VWAP, BargainIndex, ExponentialIndex, TableView	0.09
VWAP, BargainIndex, ExponentialIndex, TableView, TcpSource	0.09
TcpSource	0.12
TcpSource, TableView	0.08

Table 1: SPPL planner response times, for a sample domain.

Table 1 illustrates the performance of our SPPL planner in MARIO domains. We have measured planning times for several selected goals within a sample application domain (i.e., stock price analysis). Typically, the users will refine goals by selecting one tag a time from possible refinements, starting with an empty goal. In Table 1 we show planning times for two refinement sessions. The SPPL planner was running on a 64-bit Linux node with a quad-core 2.93Ghz Intel Xeon processor with 32GB RAM, however only a single core was used, and the memory usage was below 2GB.

Cascade IDE: A Development Environment for Composition Domains

Cascade IDE is a set of Eclipse (Eclipse Foundation) plugins that developers can use to create and maintain Cascade descriptions of application composition domains for MARIO. The IDE can also import existing stream processing code and generate Cascade descriptions. For example, a stream processing operator invocation implementing exponential bargain index computation can be imported, and its output can be annotated with *ExponentialIndex* tag.

The integration with Eclipse has made it possible to develop a full-featured Cascade editor with syntax highlighting, auto-completion and refactoring. Refactoring allows, for example, to rename a tag, In addition to the editor for Cascade, the IDE includes editors for Web UI configuration and tag relationships.

The planning domains described in Cascade can be deployed directly from the IDE to a MARIO server. Then, double-clicking on a server entry in the IDE launches a browser showing the Web interface. This automation significantly reduces the time spent on routine tasks during the development and testing of application composition domains.

Among many challenges associated with making automated software composition practical, perhaps the most significant is the need to create tools that help developers simultaneously debug a large family of applications generated from a single Cascade project. To address this problem, we have integrated automated test generation and execution techniques with our Cascade IDE (Winbladh and Ranganathan 2011).

Conclusion

We have developed MARIO, a goal-driven tool for automated composition of stream processing applications. This tool allows business users to create applications for their goals without programming. We have also built an Eclipse-based IDE for developers, which allows to describe composable components, such as stream processing operators and data sources, as well as composition patterns, and to make these components and patterns available for application composition by the business users. Our tools have been used in a customer pilot since 2009, and are currently being extended to support other target platforms in addition to IBM InfoSphere Streams.

Acknowledgements and Credits

This work, which has begun eight years ago, would not be possible without the support of our customers, and the contributions from our colleagues and interns at IBM Research. Our former colleagues and team members Zhen Liu, Eric Bouillet, and Hanhua Feng have each contributed to MARIO very significantly. The authors also thank Kristina Winbladh, Shirin Sohrabi and Genady Grabarnik for their contributions. We thank Nagui Halim for continued discussions and valuable feedback. Finally, we thank the anonymous reviewers of our papers who helped improve our work.

References

Bouillet, E.; Feblowitz, M.; Liu, Z.; Ranganathan, A.; and Riabov, A. 2008. A tag-based approach for the design and composition of information processing applications. In *OOPSLA*, 585–602.

Eclipse Foundation Eclipse Project. http://eclipse.org.

IBM InfoSphere Streams http://www.ibm.com /software/data/infosphere/streams/.

Liu, Z.; Ranganathan, A.; and Riabov, A. 2007. A planning approach for message-oriented semantic web service composition. In *AAAI*, 1389–1394.

Ranganathan, A.; Riabov, A.; and Udrea, O. 2009. Mashup-based information retrieval for domain experts. In *CIKM*, 711–720.

Riabov, A., and Liu, Z. 2006. Scalable planning for distributed stream processing systems. In *ICAPS*.

Traverso, P., and Pistore, M. 2004. Automated composition of semantic web services into executable processes. In *ISWC04*.

Winbladh, K., and Ranganathan, A. 2011. Evaluating test selection strategies for end-user specified flow-based applications. In *ASE*, 400–403.

FlowOpt: Bridging the Gap Between Optimization Technology and Manufacturing Planners

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

[1] Charles University, Faculty of Mathematics and Physics, Malostranské nám. 25, Praha, Czech Republic

[2] Entellexi Ltd., National Technology Park, Limerick, Ireland

[*] bartak@ktiml.mff.cuni.cz (contact e-mail)

Abstract

FlowOpt is an integrated collection of tools for workflow optimization in production environments. It was developed as a demonstration of advancements in the areas of modeling and optimization with the focus on simplifying the usage of the technology for end customers. The system consists of several interconnected modules. First, the user visually models a workflow describing the production of some item. Then the user specifies which items and how many of them should be produced (order management) and the system automatically generates a production schedule. This schedule is then visualized in the form of a Gantt chart where the user can arbitrarily modify the schedule. Finally, the system can analyze the schedule and suggest some improvements such as buying a new machine. Constraint satisfaction technology is the solving engine behind these modules.

Introduction

One of the biggest problems of today`s advanced technology is its limited accessibility to users working in a given domain but not necessarily being experts in the used technology. Apple's iPhone is a great example how advanced technology can be made accessible to regular users. With the tradeoff of slightly limited functionality, it provides a user interface to very advanced techniques such as Q&A (question and answering) that anyone can immediately use without the hassle of long training.

FlowOpt is a system that attempts to address the above problem and bridges the gap between advanced optimization technology developed at universities and practitioners from production planning. In particular FlowOpt is targeted to production planning in Small and Medium Enterprises. It covers modeling, optimizing, visualizing, and analyzing production processes in a streamlined feature-rich environment. FlowOpt is a student software development project at Charles University in Prague (Czech Republic). The software itself is a collection of closely interconnected modules that are plugged into the enterprise performance optimization system MAK€ from Entellexi Ltd. (Ireland).

FlowOpt Functionality

FlowOpt covers almost the complete production-planning cycle. It allows users to describe visually and interactively the process of producing any item in the form of a nested workflow with alternatives. After specifying what and how many items should be produced, the system generates a production plan taking in account the existing resources in the factory. The plan is visualized in the form of a Gantt chart that uses information about workflows and allows users to arbitrarily modify the plan by selecting alternative

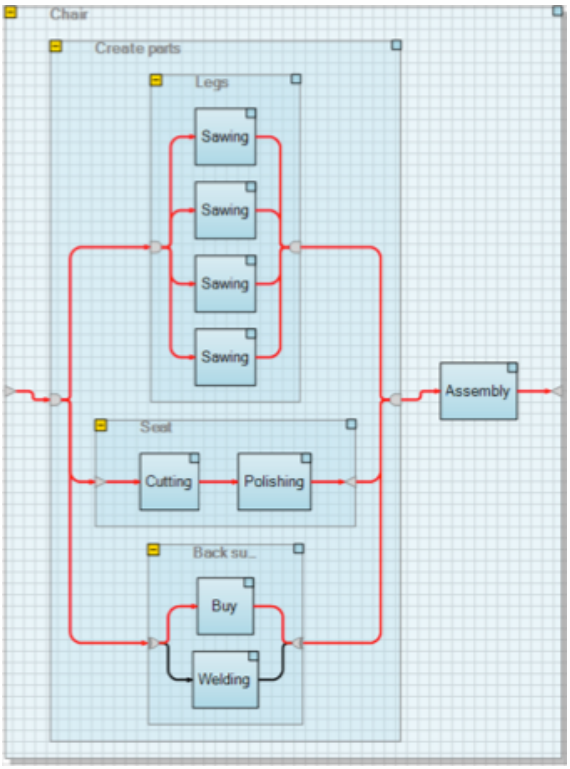


Figure 1. Visualization of a nested workflow in the FlowOpt Workflow Editor (from top to down there are parallel, serial, and alternative decompositions)

processes or allocating activities to different times or resources. Finally, the schedule can be analyzed, the bottleneck parts are highlighted and some improvements are suggested to the user. We will now introduce the functionality of individual modules.

Workflow Editor allows users to create and modify workflows in a visual way. We use the concept of nested workflows that are built by decomposing the top task until the primitive tasks are obtained (Barták and Čepek 2008). Three types of decompositions are supported: 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* (Figure 1). The final primitive tasks are then filled with activities defined in the MAK€ system (Barták, Sheahan, Sheahan 2012); each activity has a given duration and a set of unary resources necessary for its processing. The workflow can be built in the top-down way by decomposing the tasks or in the bottom-up way by composing the tasks. In practice the user can combine both approaches by decomposing any task or composing a collection of tasks to a form a new task that is then placed to the hierarchical structure of the root task. In addition to the core nested structure, the user can also specify extra binary constraints between the tasks such as precedence relations, temporal synchronizations (start-start, end-start, end-end), or causal relations (mutex, equivalence, and implication). Everything is done using an intuitive drag-and-drop approach. The system also supports import of foreign workflows and it has the function of fully automated verification of workflows (Rovenský 2011). The goal of verification is to find structural problems, namely to find tasks that cannot be part of any valid process due to workflow constraints. Figure 2 gives an example of output of workflow verification with highlighted flaws.

After defining the workflows for all items, this is the *modeling stage*, it is possible to start generating production plans. This is as easy as selecting the required items (workflows) in the **Order Manager**, specifying their quantities and required delivery date and starting the **Optimizer** by pressing a single button in GUI. The data about workflows, activities, and resources are automatically converted to the scheduling model and the system produces a schedule that is a selection of tasks from the workflows (if there are alternatives) and their allocation to resources and time. The Optimizer attempts to optimize both earliness and lateness costs that are derived from the delivery dates. Currently, the Optimizer supports only unary resources.

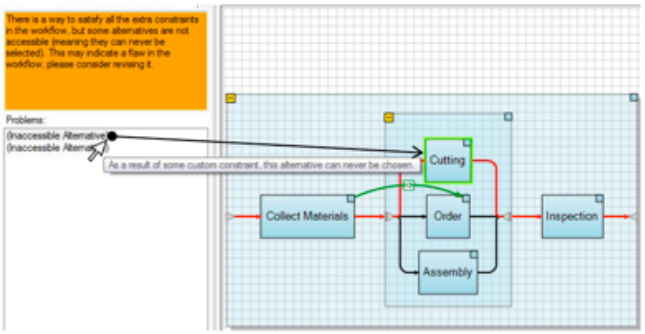


Figure 2. Highlighting the found flaws after workflow verification. The system shows all tasks that cannot be part of valid processes.

The generated schedule (production plan) can by visualized in the **Gantt Viewer**. This module provides both traditional views of the schedule, namely the task-oriented and resource-oriented views. Because the Gantt Viewer has full access to the workflow specification, it can also visualize the alternatives that were not selected by the Optimizer. The Gantt Viewer allows users to modify any aspect of the production plan using the drag-and-drop techniques. The user can move activities to different times and resources and change their duration. It is even possible to select another alternative than that one suggested by the Optimizer. Because the Gantt Viewer is aware about all the constraints originating from the workflow specification, it can also highlight violation of any of these constraints (Figure 3). Even more, the Gantt Viewer can automatically repair all flaws that were introduced to the schedule by the user's modifications (Barták and Skalický 2009).

The final module is **Analyzer** that is responsible for suggesting improvements of the production process. The Analyzer first finds bottlenecks in a given schedule, for example an overloaded resource. For each bottleneck, the analyzer suggests how to resolve it – this could be by buying a new resource or by decreasing the duration of certain activities (for example by staff training). The Optimizer then evaluates each such improvement and finds possible relations between the improvements, for example that applying two improvements together has more benefits than the sum of contributions of individual improvements. Finally the system selects a set of improvements such that their combination brings the best overall improvement of the production process under given constraints such as a limited budged to realize the improvements.

Technology Inside

The FlowOpt system is unique combination of modeling and optimization techniques. It is build around the concept of Nested Temporal Networks with Alternatives (Barták and Čepek 2008) that were suggested as a model of production workflows with hierarchical structure and alternative processes. In FlowOpt this concept was slightly modified and extended with additional constraints. These constraints may introduce flaws to the nested structure (for example a cycle) and hence novel verification techniques for

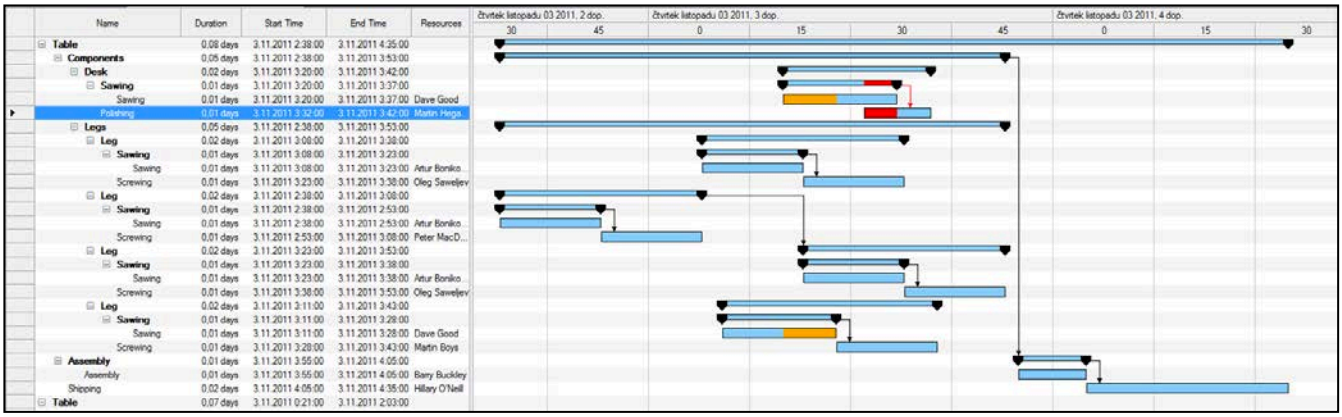


Figure 3. The task view of the FlowOpt Gantt Viewer with two highlighted constraints that are violated (exceeded capacity of resource Dave Good and broken precedence constraint).

workflows were proposed and implemented (Rovenský 2011). The general verification technique is based on modeling the problem as a constraint satisfaction problem and using advanced temporal reasoning techniques, namely IFPC algorithm (Planken 2008) to validate that there exists a feasible process for each task in the workflow. The information about workflows is combined with data about activities and resources to automatically build a scheduling model (Barták et al. 2010). Again, we use constraint satisfaction techniques to solve the scheduling problem; in particular, ILOG CP Optimizer is used to generate optimal schedules (Laborie 2009). The schedule is visualized in the form of a Gantt chart where the user can modify it. The Gantt viewer highlights constraints violated by user intervention, but it can also automatically repair these constraints using a technique of shifting activities locally in time (Barták and Skalický 2009). Again, constraint satisfaction techniques and IFPC algorithm (Planken 2008) are used in background. Finally, the Analyzer uses the idea of critical paths to discover weak parts of the schedule. Currently it uses ad-hoc rules to suggest some improvements (overloaded resource → buy a new resource). The improvements are then applied to the scheduling model and the Optimizer generates a new schedule whose cost is used to evaluate the improvement. Some interactions between the possible improvements are also discovered during this process. For example, the Analyzer can find that one improvement strengthens another improvement. From the set of possible improvements, a subset with the best overall cost is selected by using the techniques of project portfolio optimization. Again, the problem is modeled as a constraint satisfaction problem and ILOG CP Optimizer is used to solve it.

Demonstration Description

The complete process of generating a production plan will be demonstrated. First, we will present the design process of modeling nested workflows using decomposition and composition of tasks. We will also add extra constraints that introduce flaws to the workflow and then we will demonstrate the workflow verification procedure and its outputs. The schedule will be generated in real time and then the visualization capabilities of the Gantt Viewer will be presented. In particular we will show how the schedule can be modified and how the system can automatically repair the violated constraints. Finally, we will present the Analyzer, namely finding the bottlenecks, proposing and evaluating improvements, and selecting the best subset of improvements.

Acknowledgments. The research and development of the FlowOpt system was supported by the Czech Science Foundation under the contract P202/10/1188 and by EU Funding Scheme Research for the benefit of SMEs under the project ValuePOLE (contract 222218).

References

Barták, R. and Čepek, O.; 2008. Nested Precedence Networks with Alternatives: Recognition, Tractability, and Models. In *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*, **21**(1), 121-132, Springer Verlag.

Barták, R.; Sheahan C.; Sheahan, A.; 2012. MAK€ – A System for Modelling, Optimising, and Analyzing Production in Small and Medium Enterprises. In *Proceedings of 38th International Conference on Current Trends in Theory and Practice of Computer Science* (SOFSEM). LNCS 7147, Springer Verlag, pp. 600-611,

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 Applications*, Dublin, Ireland, pp. 507-517.

Laborie, P.; 2009. IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (CP-AI-OR), LNCS 5546, Springer Verlag, pp. 148-162.

Planken, L.R.; 2008. *New Algorithms for the Simple Temporal Problem*. Master Thesis, Delft University of Technology.

Rovenský, V.; 2011. *Workflow Modeling*. Master Thesis, Charles University in Prague, 2011.

itSIMPLE4.0: Enhancing the Modeling Experience of Planning Problems

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

Introduction

The itSIMPLE project (Vaquero et al. 2007; 2009) is a research effort to develop a reliable knowledge engineering (KE) environment to support the design of AI planning applications. Unlike other KE tools for AI planning, itSIMPLE focuses on the initial phases of a disciplined design cycle, facilitating the transition of requirements to formal specifications. Requirements are gathered and modeled using *Unified Modeling Language* (UML) (OMG 2005) to specify, visualize, modify, construct and document domains or artifacts, in an object-oriented approach. A second representation, Petri Nets (PN) (Rozemberg and Engelfriet 1998; Murata 1989), is automatically generated from the UML model and used to analyze dynamic aspects of the requirements such as deadlocks and invariants. A third representation, Planning Domain Description Language (PDDL) (Gerevini and Long 2006), is also automatically generated in order to input the planning domain and instance into an automated planner.

itSIMPLE's framework and translators reduce the gap between real planning applications which are seldom represented directly in PDDL and state-of-the-art AI planners. itSIMPLE is an open-source, Java-based system that has been applied to several real planning applications since 2005, including petroleum supply port management (Sette et al. 2008), project management (Udo et al. 2008), advanced manufacturing (Vaquero et al. 2006), information systems, and intelligent logistics systems.

In this paper we describe the new features implemented in version 4.0 of the itSIMPLE system. These new features aim to enhance the modeling experience and provide designers with extra tools to facilitate the model creation process, from knowledge acquisition and modeling to plan generation and analysis.

In what follows, we give a brief overview of the design environment of itSIMPLE, sketching each available phase of a design process, and the framework that integrates a set of languages and formalisms used during the design process. Next, we describe the new features of the tool (version 4.0) and some future directions.

The Design Environment: Towards a Disciplined Modeling Process for Planning

A completely formal design process is not possible since it starts, by definition, with non-formalized, and perhaps tacit, knowledge. itSIMPLE is divided into four primary phases, which may be re-entered multiple times in an iterative fashion and that define a design process which aims to capture the essence of non-formalized knowledge and transform it into a formal description of the planning domain. Each phase is described below.

Requirements Elicitation and Modeling

Any planning system is embedded in a real environment, that is, a myriad of "non-system" tools, objects, people, and processes with which it must interact. It is necessary to have a detailed model of this domain environment and it is important that this model be developed independently of the planning system (McDermott 1981) based on the concept of a *work domain* from Cognitive Design (Naikar, Hopcroft, and Moylan 2005).

In itSIMPLE, requirements and knowledge are gathered and modeled in an object-oriented fashion using a suite of UML diagrams: class, state-machine, timing, and object diagrams. The UML diagrams are used to represent the main aspects of the domain objects in the work domain and planning problem such as static information (objects and agents, defined by classes and relations), dynamic information (state transitions and features changed by actions) and problem instance description (snapshots of objects and relationships describing the initial state, goal state and desirable intermediate states for instance).

Domain Analysis

The Domain Analysis phase is based on static information analysis and validation of dynamics using a state-transition approach. Static analysis is performed by creating snapshots and possible scenarios (using object diagrams) based on the class diagrams and all constraints defined on them (Vaquero et al. 2007). Dynamic analysis is performed by simulation of Petri Nets created from UML models.

Plan Development

After modeling and analyzing the domain, itSIMPLE uses PDDL to communicate the model to solvers, including Metric-FF, FF, SGPlan, MIPS-xxl, LPG-TD, LPG, hspsp, SATPlan, Plan-A, Blackbox, MaxPlan, LPRPG, POPF, and Marvin. Designers can run these planners and obtain the resulting plans to be analyzed, all in the same environment. This feature gives itSIMPLE a significant flexibility to exploit recent advances in solver technology.

Plan Analysis

itSIMPLE provides some functionality for plan analysis, including plan visualization and plan simulation. A plan visualization and simulation is provided by a functionality called "Movie Maker" (Vaquero et al. 2007). This functionality captures the model of a domain and the plan specification in PDDL and shows the simulation of interactions between the plan and the domain through a sequence of object diagrams. Another important functionality is to be able to analyze plans according to domain variables, called "Variable Tracking".

Language Framework

The language framework of itSIMPLE was designed to be flexible and open, allowing different languages to be added. In order to integrate the languages and phases noted above, the environment uses extensible markup language (XML) (Bray et al. 2004) as a core meta-language. More details about the translation processes can be found in (Vaquero et al. 2009).

The New Features and Improvements

In this section we describe the new developed features in itSIMPLE4.0.

PDDL from Start to End, If Needed

In order to support planning experts, we have expanded itSIMPLE's framework to allow the creation of PDDL models from scratch. We have designed an initial PDDL editor so that user can load, edit and save PDDL files with domain and problems descriptions.

PDDL files from IPC or others sources can be easily loaded in itSIMPLE. The tool imports and automatically separates the elements of domain and problems contained in the file. The user can directly edit the domain model and problem description in the itSIMPLE interface. The PDDL editor in itSIMPLE also allows the user incorporate new actions, predicates, goals, initial states and many others PDDL components by using templates. The editor has a syntax highlighting that differentiates language elements by color. After editing, users can use any planner integrated with the itSIMPLE tool and analyze the generated plan.

Using Modeling Patterns

In domain-independent planning, some researchers have investigated the identification of common structures in the domain model in order to trigger a more appropriated planning strategy (Long and Fox 2000; Clark 2001). Long & Fox (2000) studied the detection of patterns (model symmetry and generic types) in the knowledge model, focusing on enhancing the performance of automated planners (e.g., STAN) by exploiting specialized techniques when particular structures are identified. Patterns have been found in several planning problems that have *transportation* (Long and Fox 2000) or *construction* (Clark 2001) characteristics. In (Long and Fox 2000), the authors characterize the main types existing in transportation problems, including, for instance, *mobile* (types of objects that move on a map), *location*, *portable* (type of object that are carried by mobiles on a map) and a hierarchy of other mobile-related generic types (Long and Fox 2000). Simpson *et al.* (2002) have made these patterns (generic types) available in the modeling tool GIPO for domain knowledge designers. Their work has provided an initial pattern language for AI planning problems for the mobile-related patterns. To our knowledge, however, none of this work has provided a definition, description format or catalog of design patterns for AI planning. Our work in this direction can be seen as a continuation of the work done by Simpson *et al.* (2002). However, our long-term goal is to (1) provide such a definition and description format of design patterns for AI Planning in a object-oriented fashion and (2) propose an initial design patterns catalog.

As a step toward our goal to provide modeling patterns to users, we have designed an initial (small) set of patterns and made them available for the user in UML. We provide a description of the intention of the pattern, scenarios, applicability, and the model representation using UML. We followed a simple approach on this version of the system in which users can import a pattern as a predefined set of UML diagrams (classes, constraints and action definitions). Based on previous work and analysis of benchmark and real planning problems we provide the following initial set of patterns:

- *Move & Reach*: a basic pattern that can be used to represent objects that can move and must reach certain positions or locations on a map. The pattern encapsulates the behavior of two classes of objects (roles): Mobile (agent) and Location. The common behavior of mobile types has already been observed and defined in (Long and Fox 2000). Here we are representing them in a object-oriented fashion with UML.
- *Transportation*: a pattern that represents agent objects (carriers) that can carry cargo items (portables) between locations on a map. The pattern encapsulates the behavior of the classes Carrier (agent), Cargo and Location. The common behavior of carriers and portable types has also been observed and defined in (Long and Fox 2000).
- *Stack & Place*: a pattern that can be used to represent an agent object that can pile up, fit and organize item objects on a surface or in a container. The pattern encapsulates the behavior of the classes Stacker (agent), Item and Surface (e.g., grid).
- *Assembling*: a pattern representing agent objects that must compose or decompose parts to create structures or other composed parts. Such (dis)assembling process has to fol-

low an order. The pattern encapsulates the behavior of the classes Assembler (agent) and Part.

Time-based Models

We have recently (since version 3.5) added features to itSIMPLE to allow the representation of some time constraints. The tool provides timing diagrams to describe how (boolean) properties of objects change during the execution of an action, i.e., defining if a property becomes true at the beginning or at the end of a durative action. These diagrams are used to translate the conditions and effects of the actions to PDDL while assigning the right temporal operator to them (e.g., *at start*, *at end*, *over all*) (Fox and Long 2003). However, the tool did not cover the spectrum of timing constraints expressible in PDDL. In the new version of itSIMPLE, we have refined the way users input time constraints. When using a timing diagram, users are now able to specify whether the diagram represents the effect or the precondition of the action (the system then translates the conditions to PDDL with the right temporal operator). The use of timing diagrams in the tool is still restricted to an action horizon (as opposed to multiple actions or a series of actions) and to boolean properties. However, we have made another extension to overcome the latter problem and to open ways to a more elaborated definition of time constraints in the actions' conditions and effects. Since users define pre- and post-conditions with the Object Constraint Language (OCL) (OMG 2003) in itSIMPLE, we let them index, or annotate, their OCL sentences with the desired temporal interval (e.g., [t1,t2] or (t1,t2]). For example, users can annotate a precondition sentence such as '*truck.at = loc and pkg.in = truck*' with the temporal interval [0,10], meaning that the condition must be true from time point 0 to 10 after an action has started. It is also possible to use reserved words such as *start* and *end* in the interval (e.g., [*start*,*start*] or [*start*,*end*] or [*end*,*end*]) to annotate any sentence in the precondition and effect (note that a precondition annotated with (start,end) or [start,end] would be translated to PDDL using the *over all* temporal operator). This extension is just one more step to translating timing constraints to PDDL. We are working on other approaches, such as timelines, to provide more modeling capabilities for time-based models.

Wizards for State Creation

We have been designing features to facilitate the task of creating large scale problem instances using UML object diagrams. We have developed wizards that can reduce the time spent creating the initial state, goal state, or any intermediary preferable state as snapshots. In many cases, users have to specify and input several pieces of data, for example about the distance between locations. Depending on the number of locations the model contains, it can be time consuming to put this information in the model. itSIMPLE allows the user to import this information from a file or even link the file in the model so the tool can use it when translating to PDDL and sending the model to a planner. The file format is currently limited to text files with each record in each line. However, we plan to expand this idea to cover input data from other types of files and chiefly from databases: users provide the queries and the database access so the tool can take care of retrieving the information from the database.

Creating associations among several objects in a diagram can also be tedious. We have designed a wizard that allows users to select a group of objects and associate them in different ways. For example, users can associate all of them, associate just the neighbors, just the object in the right, just in the left, and so forth. We have also designed wizards that can create a predefined network of objects. For example, users can create grids of locations or places by providing the size of the grid, the class that represents the nodes in the grid, the association that defines the edges of the network, and the list of names of the generated objects (if necessary). In addition, when selecting more the one object of the same type it is possible to set their attribute values once, without having to define them individually.

Exchanging and Sharing Experimental Setups

As described in previous sections, itSIMPLE is integrated with several automated planners. Users can set up experiments with the available planners, simulating what is done in IPC. In previous versions of the system, designers were able to select their favorite planners and run them over a set of planning problem instances or even over several instances of different domains. In the current version, users are able to store the setup of an experiment in a XML file and share it with others. A stored experimental setup can be used to run the same experiment in different machines. For example, one can design his/her experiment on a laptop, export the setup and run it on a server later. It can also be exchanged and reused among researches and designers. Such a setup stores the information about what planners must be executed, in which order, what needs to be recorded (e.g., runtime, metrics), and the timeouts. Initially, users need to have itSIMPLE installed in the machines that the experiment will run.

Future Directions

In the itSIMPLE project, we aim to provide the ultimate tool for modeling, testing, analyzing and deploying AI planning domain models. itSIMPLE will be extended to deal with semantic analysis of UML and PDDL models, cross-validation among model components and model dynamic simulation through Petri Nets in the future. Many items that lead the tool to encompass all of these features have been implemented, but there are many more to develop. itSIMPLE must be improved with a more complete and accurate time-based modeling, as well as allowing the user to model HTN domains and domains with probabilistic features. HTN domains can be easily modeled by using activity diagrams which are native to UML. Probabilistic features demand an extension of UML: a more detailed analysis of the viability of this direction is needed.

Since itSIMPLE works with UML, it is natural to think that it can deal with design patterns to help the user to model domains by capturing some past solutions to improve the current model. Initial design patterns have been already incorporated in itSIMPLE tool. However, much work is still

needed to reach a full use of design patterns in UML planning models.

itSIMPLE has been remodeled to become more feasible and useful. Some new ways to model and manipulate the planning environment and applications in UML, PDDL or Petri Nets have changed and will be changing more in the near future. All of these modifications aim to provide a tool with more usability and user-friendly features.

Conclusion

itSIMPLE has been developed since 2004 and presented since the first ICKEPS competition in 2005. Since then, we have been working on eliminating remaining gaps between practical applications and planning systems. Many features have been implemented over the last eight years and itSIMPLE is finally becoming a tool that can allow researchers, industrial users, students and other stakeholders to take advantage of recent planning development. The new features described in this paper show that itSIMPLE has reached a mature level with a stable version of UML modeling, domain analysis and planning simulations. It is a user-friendly tool not only for beginners, students and non-planning experts, but also for planning experts.

References

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2004. Extensible Markup Language (XML) 1.0 (Third Edition). Technical report.

Clark, M. 2001. Construction domains: A generic type solved. In *Proceedings of the 20th U.K. Planning and Scheduling Workshop*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension of pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research (JAIR)* 20:61–124.

Gerevini, A., and Long, D. 2006. Preferences and soft constraints in pddl3. In Gerevini, A., and Long, D., eds., *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, 46–53.

Long, D., and Fox, M. 2000. Automatic Synthesis and use of Generic Types in Planning. In *Artificial Intelligence Planning and Scheduling AIPS-00*, 196–205. Breckenridge, CO: AAAI Press.

McDermott, J. 1981. Domain knowledge and the design process. In *DAC '81: Proceedings of the 18th conference on Design automation*, 580–588. Piscataway, NJ, USA: IEEE Press.

Murata, T. 1989. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, 541–580.

Naikar, N.; Hopcroft, R.; and Moylan, A. 2005. Work domain analysis: Theoretical Concepts and Methodology. Technical report.

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

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

Rozemberg, G., and Engelfriet, J. 1998. Elementary net systems. *Lecture Notes in Computer Science* 1491:12–121. Springer.

Sette, F. M.; Vaquero, T. S.; Park, S. W.; and Silva, J. R. 2008. Are automated planners up to solve real problems? In *Proceedings of the 17th World Congress The International Federation of Automatic Control (IFAC'08), Seoul, Korea*, 15817–15824.

Udo, M.; Vaquero, T. S.; Silva, J. R.; and Tonidandel, F. 2008. Lean software development domain. In *Proceedings of ICAPS 2008 Scheduling and Planning Application workshop. Sydney, Australia.*

Vaquero, T. S.; Tonidandel, F.; Barros, L. N.; and Silva, J. R. 2006. On the use of uml.p for modeling a real application as a planning problem. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 434–437.

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

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

Inspect, Edit and Debug PDDL Documents: Simply and Efficiently with PDDL Studio

Tomas Plch[1], Miroslav Chomut[2], Cyril Brom[3], Roman Barták[4]
Faculty of Mathematics and Physics, Charles University in Prague
[1]tomas.plch@gmail.com [2]chmirko@gmail. com [3]brom@ksvi.mff.cuni.cz [4]bartak@ktiml.mff.cuni.cz

Abstract

The Planning Domain Definition Language (PDDL) represents a standard for definitions of planning domains and problems. Researchers and designers often make semantic and syntax errors due to the language's complexity. At the same time, it is hard to read and work with larger documents in PDDL. We have developed a tool called *PDDL Studio*, which is aimed at aiding in creation and inspection of PDDL documents. The editor's main features are: 1) PDDL parser capable of localizing syntax and semantic errors, 2) PDDL syntax highlighting, 3) context sensitive code completion and hints - similar to Microsoft's *IntelliSense* for declarative languages, 4) code collapsing, 5) PDDL document management, and 6) planner integration. Our PDDL Editor also features a PDDL Parser tool, which can be used as a standalone parser for other projects.

Introduction

The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) represents a standard for creating definitions of planning domains and problems and is utilized as input for various planners e.g. JSHOP, JSHOP2 (Nau et al. 1999), BlackBox (Kautz and Selman 1998), Metric-FF (Hoffman 2003).

The current practice is to create PDDL documents either by hand via simple editing tools e.g. Notepad++, or via tools and languages for domain knowledge and characteristics specification (i.e. knowledge engineering). To name a few of such tools: itSimple (Vaquero et al. 2009) utilizes the combination of graphical UML specification and XML, GIPO IV (Simpson 2007) uses custom diagram notation, and ViTAPlan (Vrakas and Vlahavas 2003) for domain and problem visualization. An extensive study of the various tools and approaches can be found in (Vaquero et al. 2011). It is noteworthy that there is a multitude of tools aimed at verification of PDDL like the VAL tool (Howey et al. 2004) or PDver (Raimondi et al. 2009).

At some point during the process of knowledge engineering for planning and scheduling, the need for directly inspecting and editing of PDDL documents often occurs. Regardless of whether these documents are created automatically or manually, they often are large and complex, thus being hard to inspect.

The imperative programming language community (i.e. utilizing languages like C++ and Java) has a wide range of Integrated Development Environments (IDEs) e.g. Visual Studio 2010 (VS10), NetBeans, and Eclipse. Various functionalities (e.g. syntax highlighting, on the fly syntax checking, code completion and contextual hints), help programmers to develop in a faster and more convenient manner. The planning community lacks such an integrated tool (i.e. editor) for PDDL. With this motivation in mind, we created our *PDDL Studio* application, which is aimed at bringing the imperative programming culture of editing source code to the planning community. In this paper we overview main features of PDDL Studio.

PDDL Studio

Our project, PDDL Studio (Figure 1), is focused on providing a simple editing IDE. The application itself is written in C++ and is designed with portability in mind, utilizing portable technologies like the Flex Lexical Parser (Paxson 2008), Bison parser generator (GNU 2011) and the Qt framework (Qt Project 1992) for visual representation.

We identified a broad range of necessary capabilities, which are present in most of applications like VS10, NetBeans, Eclipse and can be applied to the PDDL language:

- *Project management* – creation and management of documents
- *Syntax error detection* – interactive localization and identification of PDDL syntax errors
- *Syntax highlighting* – coloration of language elements

- *Semantic error detection* – detection of simple semantic errors
- *Context sensitive code completion* – providing hints to the user what to write based on the current context in the document's input
- *Code collapsing* – portions of the code can be *collapsed* to provide better readability
- *XML import/export* – creating an XML variant of the PDDL document
- *Planner integration* – the ability to execute a planner with the PDDL documents as input
- *Common Editor Features* – line numbering and bracket matching

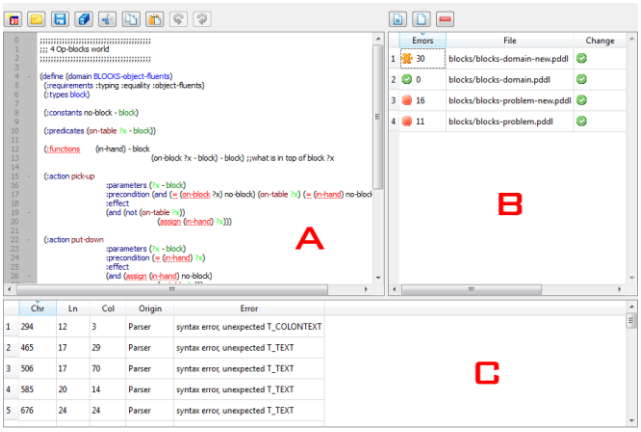


Figure 1: PDDL Editor window with Editing Window (A) having highlighted syntax, Interactive Error Report (C) and Project Manager (B) with additional information about file status (number of errors, save status etc.)

The remainder of this section is focused on describing the realization of the outlined capabilities in PDDL Studio. The main aim of the realization is to provide intuitive and easy to use features that would facilitate the task of creating PDDL encodings.

Project management

Presently, the *Project Management* in PDDL Studio is rather simplistic, providing only the capability to create projects, add and remove files from the project. The Project Manager also provides additional information on project files (e.g. present error counts, change status etc.).

Syntax error detection

Syntax error detection is one of the most important features of the PDDL Studio. We created our own dedicated PDDL parser build upon Flex and Bison (presently supporting PDDL 1.7), making the project more portable. The parser is designed to be used independently from the PDDL Studio's code as a library or C++ code. The parsing process creates a *tree-like representation* of the PDDL document's elements. Our testing domains of 1000 lines can be parsed below 100 ms on a standard Dual Core notebook processor at 1.8GHz utilizing one core.

The error detection mechanism is executed on the fly during editing of the PDDL document. If the user only views the file, the detection mechanism is suspended. If the user inserts or removes text, the detection mechanism is prepared for execution after a predefined period of inactivity – i.e. the user stopped typing. The parsing mechanism can be further suspended if the user resumes typing. This allows providing smoother error detection, avoid over-consuming resources, and avoid bothering the user with false positives on syntax errors.

The error detection module provides a list of errors presented as an interactive error table and underlines the found errors in the document. It can be seen in Figure 1 (C). When accessed (e.g. double click on an error's row), the editor points directly to the detected error. In respect to the PDDL specification, we can localize misspelled or missing keywords. When a mandatory keyword is missing, we also identify which keyword is missing, thus providing a hint to the user.

Syntax Highlighting

Based on our experience with various programming IDEs, we perceive the syntax highlighting as one of the most important aids when inspecting any code. It is a key feature included in every usable IDE tool since colored fonts were available. It vastly improves the ability for the developer to read code and distinguish language elements – i.e. variables, types, functions, predicates.

We use the tree-like structure provided by our parser to identify language elements and assign colors to the resulting editable text. The result can be seen in Figure 1 (A). The user can set his preferred colors for the edited text and the following language constructs:

- problem and domain names
- variable names
- detected errors – these are underlined for better display and this color overrides the color of any other element
- PDDL keywords – e.g. *requirements, predicates* etc.
- predicate names
- type names
- highlighted brackets – pairs of brackets are highlighted when the user points the editing cursor at one of them

Semantic error detection

The PDDL Studio is also capable of detecting semantic errors in respect to the language specification and the provided requirements (e.g. *Disjunctive-Preconditions*). The parsed tree-like structure is used to determine where these errors are located in the PDDL documents and what

their nature is. This information is provided within the error table (Figure 1(C)). We can detect the following semantic errors:

- Use of non-existent types – when the user misspells a type which is not present in the *type* declaration. This is checked based on the *typed* requirement.
- Use of non-existent predicates – when a predicate is misspelled or not defined correctly.
- Inconsistent use of predicates parameters – when a predicate is used with wrong parameter types.
- Inconsistent use in respect to domain requirements.

Context sensitive code completion

Code completion is an important feature currently included in every major IDE. The basic idea is to provide the user with hints based on the current scope (e.g. available functions, keywords etc.) while editing the document. This feature takes the burden of the user to avoid errors i.e. typing errors and syntactic and semantic errors. It also provides a speed-up for development.

Our code completion is context sensitive – based on the current edited portion of the PDDL document as can be seen in Figure 2.

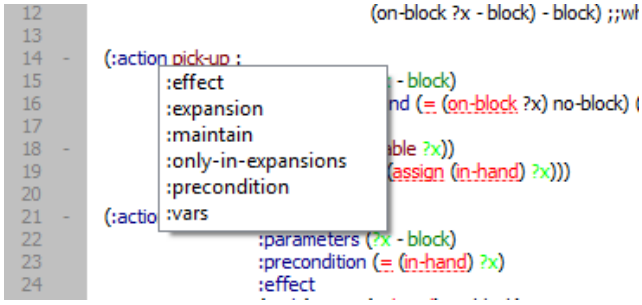


Figure 2: Context sensitive auto completion – a hint is given for a subsection of the Action element

We can provide completion hints in the following areas:

- language keywords – basic PDDL language elements
- domain specification for problems on known domains in the project
- predicates for problem initialization on known predicates in the project
- content for the requirements specification
- defined variables and parameters
- defined predicates

Code collapsing

The code collapsing feature is important for better code readability. Parts of the code – code blocks – can be hidden, because the reader does not need to read them at the moment.

Our project provides possibility to collapse automatically detected portions of code (e.g. actions, predicates etc.). The PDDL language is based on Lisp notation, therefore is full of code blocks. We presently support only high level code collapsing (Figure 3) – i.e. at the level of whole actions, predicates etc. We are working on a method to specify how deep the code collapsing should be allowed to maintain readability and limit the amount of collapse points. This context sensitive code collapsing feature is currently under development.

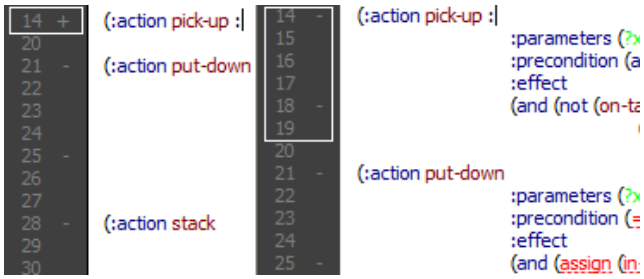


Figure 3: Code collapsing of an *action* pick-up block. The '-' sign is used to indicate collapse points, the '+' sign is used to indicate expansion points.

XML export and import

On one hand the PDDL language is hard to read and on the other hand it is hard to parse. We created our XML equivalent of the PDDL Language notation. The PDDL Studio can export and import this format for use by other applications or for better readability by other developers.

Planner integration

PDDL Studio provides the capability to integrate any planner which can be executed from a system command line – i.e. console prompt. We provide the user with our execution console which allows for project specific simple scripts (Figure 4).

These scripts are parsed and the result is executed via the system console. Various parameters can be used, e.g. current project directory, file names of PDDL files etc.

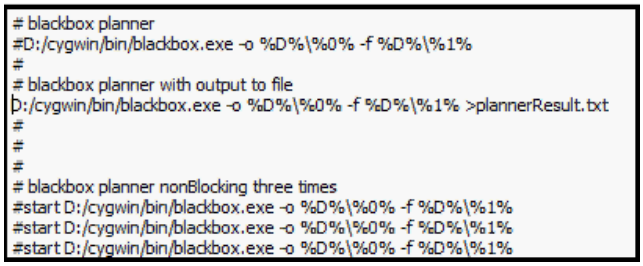


Figure 4: Planner integration using simple scripts executed in trough the system command line

Common Editor Features

The PDDL Editor also incorporates common editor features like *line numbering* and *bracket matching*. Line numbering is represented in the left portion of the screen. Bracket matching is used to identify bracket pairs by coloring them. It allows the user to detect missing brackets.

Conclusion and Future work

The PDDL Studio is a simple but powerful tool for creation and management of PDDL projects. We created it to mimic the behavior of the commonly and widely utilized IDEs like Visual Studio 2010 or Eclipse (in respect to code editing). The main features of this tool are the capability to locate and identify syntax and semantic errors in the PDDL document and provide semantic hints on code completion. The tool also provides features like syntax highlighting and code collapsing, which allow the user to read and inspect the code easily.

In the next version, we intend to integrate a more complex and capable project management system similar to the one present in VS10. We intend to extend our PDDL parser to be used with the newest version of PDDL. We also work on extending our semantic error detection and work on integrating our tool with a plan visualization and inspection tool and our own planner.

We also intend to provide the user with the capability to create custom templates for various purposes – e.g. action skeletons filled based on given or guessed parameters. We also want to provide automated on the fly indentation of documents. The user might request a view-only custom indentation of the displayed documents to suit his reading/writing style. We also want to incorporate a simple mechanism to invoke various actions (e.g. commenting of selected text portions, inserting custom text templates etc.) via user defined key combinations (e.g. Ctrl + Shift + F1).

Acknowledgement: This work was partially supported by a student grant GA UK No. 0449/2010/A-INF/MFF, by project P103/10/1287 (GA ČR) and GA UK No. 655012/2012/A-INF/MFF.

The application can be downloaded at: http://amis.mff.cuni.cz/PDDLStudio.

References

GNU Project. 2001. GNU Bison. http://www.gnu.org/software/bison (ver. 2.5 2011).

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. Journal of Artificial Intelligence Research, Volume 20, pages 291 - 341.

Howey, R., Long, D., Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. Tools with Artificial Intelligence, ICTAI 2004

Kautz, H., Selman, B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98, Pittsburgh, PA.

McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M.; Weld D., Wilkins D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

Nau, D., Cao, Y., Lotem, A., Muñoz-Avila, H. 1999. *SHOP: Simple Hierarchical Ordered Planner*. In IJCAI-99, pp. 968-973.

Paxson, V. 1987. Flex Lexical Analyzer. http://flex.sourceforge.net (ver. 2.5.35 2008).

Qt Project. 1992. Qt. http://www.qt-project.org (ver. 4.8.0 2011).

Raimondi, F., Pecheur, C., Brat , G. 2009. *PDVer, a tool to verify PDDL planning domains*. In Proceedings of ICAPS'09 Verification and Validation of Planning and Scheduling Systems.

Simpson, R. M. 2007. Structural Domain Definition using GIPO IV. In Proceedings of the Second International Competition on Knowledge Engineering for Planning and Scheduling

Vaquero, T., S., Silva, J., R., Beck, J., C. 2011. A Brief Review of Tools and Methods for Knowledge Engineering for Planning & Scheduling. In: Proceedings of the Knowledge Engineering for Planning and Scheduling (KEPS) workshop. The 21th International Conference on Automated Planning & Scheduling (ICAPS 2011). Freiburg. Germany.

Vaquero, T. S., Silva, J. R., Ferreira, M., Tonidandel, F., Beck, J. C. 2009. *From Requirements and Analysis to PDDL in itSIMPLE3.0*. In Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS 2009, 54–61.

Vrakas, D. and Vlahavas, I. 2003. ViTAPlan: A Visual Tool for Adaptive Planning, In Proceedings of the 9th Panhellenic Conference on Informatics, Thessaloniki, Greece, pp. 167-177.

Demonstration Abstract: New Developments in Real-Time Heuristic Search

Carlos Hernández
Depto. de Ingeniería Informática
Univ. Católica de la Ssma. Concepción
Concepción, Chile
chernan@ucsc.cl

Jorge A. Baier
Depto. de Ciencia de la Computación
Pontificia Univ. Católica de Chile
Santiago, Chile
jabaier@ing.puc.cl

Tansel Uras **Sven Koenig**
Department of Computer Science
University of Southern California
Los Angeles, USA
{turas,skoenig}@usc.edu

Abstract

Our demonstration consists of a poster, videos and interactive simulations of real-time heuristic search algorithms for goal-directed navigation on a priori completely or partially unknown grids. It provides a brief introduction to real-time heuristic search by describing LSS-LRTA* and RTAA*. It then illustrates a performance issue of LSS-LRTA* and RTAA* due to depressions in the h-value surface. It describes the real-time heuristic search algorithms aLSS-LRTA*, daLSS-LRTA*, aRTAA*, and daRTAA*—which address this issue—and summarizes their properties. Our demonstration also illustrates a performance issue of LSS-LRTA* and RTAA* due to performing repeated A* searches around the current cells of the agent. It describes RTBA* and TBAA*, two real-time heuristic search algorithms that address this issue, and summarizes their properties.

Motivation

Many applications require agents to act quickly in a priori completely or partially unknown environments. Examples range from autonomous cars to video games, where companies impose time limits on the order of 1 millisecond (ms) for path planning (Bulitko et al. 2011). Such time limits are insufficient for finding complete paths, and an agent thus needs to move before it has found a complete path. We use goal-directed navigation on a priori completely or partially unknown grids with blocked and unblocked cells as running example. The agent always observes the blockage status of its eight neighboring cells and has to move from a given start cell to a given goal cell by repeatedly moving to an unblocked neighboring cell. Performance is measured by the number of moves before the agent reaches the goal cell. We study an agent that uses real-time heuristic search algorithms (Korf 1990) to determine its moves. Real-time heuristic search algorithms interleave A* searches (Hart, Nilsson, and Raphael 1968) with moves. We assume that the reader is familiar with A* and the associated terminology and give two examples of real-time heuristic search algorithms for goal-directed navigation on a priori completely or partially unknown grids in the following, both of which are variants of LRTA* (Korf 1990):

- LSS-LRTA* (Koenig and Sun 2009) assumes that cells with unknown blockage status are unblocked (Zelinsky 1992; Koenig, Tovey, and Smirnov 2003). This free-space assumption allows LSS-LRTA* to perform an A* search from the current cell of the agent to the goal cell until the goal cell is about to be expanded, the open list becomes empty or a given number of cells have been expanded. If the open list becomes empty, the agent stops unsuccessfully. The states in the closed list form the local search space (LSS). LSS-LRTA* sets the h-values of all states in the closed list to the largest possible h-values that maintain the consistency of all h-values. The agent then moves along the shortest path from its current cell to a cell with the smallest f-value found by the A* search and remembers all blocked cells that it observes. If it reaches the goal cell, it stops successfully. If it observes a blocked cell on the current path or reaches the end of the path, it repeats the process.
- RTAA* (Koenig and Likhachev 2006b) is almost identical to LSS-LRTA*; the difference is that it updates the h-values faster than LSS-LRTA*. RTAA* often outperforms LSS-LRTA* even though the h-values of RTAA* are typically not as informed as the ones of LSS-LRTA*. However, this is often compensated for by RTAA* being able to expand more cells within a given time limit (Koenig and Likhachev 2006b; Hernández and Baier 2012).

In the following, we discuss new developments in real-time heuristic search that address two drawbacks of existing real-time heuristic search algorithms such as LSS-LRTA* and RTAA*, namely poor performance due to depressions in the h-value surface and due to performing repeated A* searches around the current cells of the agent. We explain these problems, sketch new real-time heuristic search algorithms that address them and describe their properties.

Heuristic Depressions

LSS-LRTA* and RTAA* have a performance issue due to heuristic depressions, that is, valleys in the h-value surface (Ishida 1992). We focus on cost-sensitive heuristic depressions (Hernández and Baier 2012), that is, connected cells in a set D that are completely and immediately surrounded by cells in a set F such that $h(s) < d(s, s') + h(s')$ for all cells

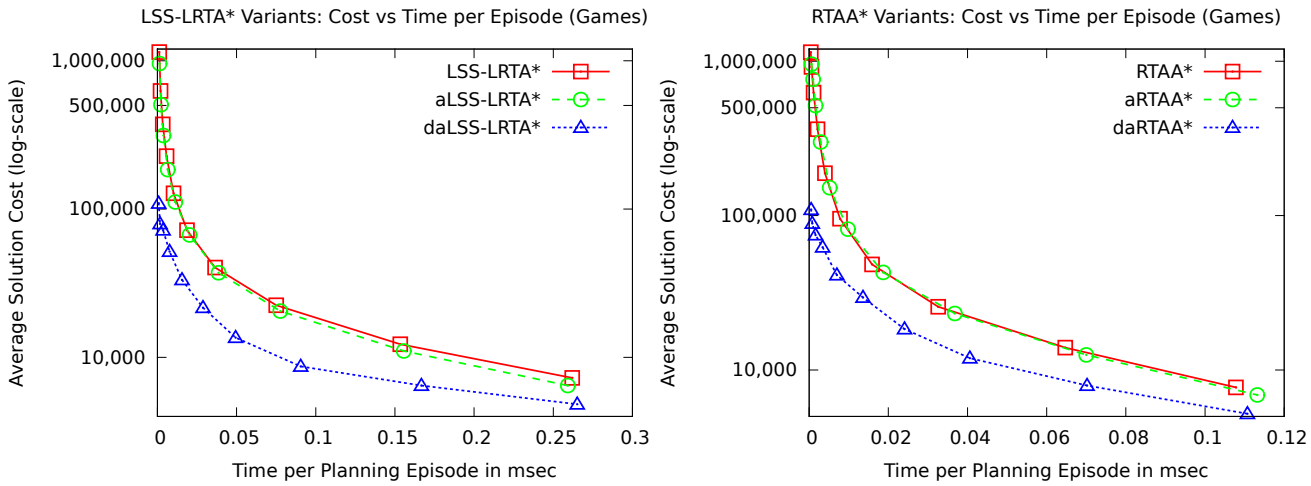


Figure 1: Evaluation of aLSS-LRTA*/daLSS-LRTA* and aRTAA*/daRTAA* on A Priori Completely Unknown Grids

$s \in D$ and $s' \in F$, where $d(s, s')$ is the distance from s to s' and $h(s)$ and $h(s')$ are the h-values of s and s', respectively. When an agent enters such a heuristic depression, it often executes many moves before it leaves the heuristic depression again. We describe real-time heuristic search algorithms for goal-directed navigation in a priori completely or partially unknown grids that address this issue by moving the agent to avoid heuristic depressions, namely aLSS-LRTA*, daLSS-LRTA*, aRTAA* and daRTAA*:

- daLSS-LRTA* (Hernández and Baier 2012) is almost identical to LSS-LRTA* and daRTAA* (Hernández and Baier 2012) is almost identical to RTAA*; the only difference is that daLSS-LRTA* and daRTAA* find a shortest path from the current cell of the agent to a cell with the smallest f-value among all cells for which the h-values have changed the least (rather than to a cell with the smallest f-value). This moves the agent to avoid heuristic depressions for the following reason: Let $\Delta(s)$ be the difference between the length of the shortest path from cell s to the goal cell and the initial h-value of s. If s is a cell close to the border of depression D and s' is a cell more in the interior of D, $\Delta(s') \geq \Delta(s)$.
- aLSS-LRTA* (Hernández and Baier 2012) is almost identical to LSS-LRTA* and aRTAA* (Hernández and Baier 2012) is almost identical to RTAA*; the only difference is that aLSS-LRTA* and aRTAA* find a shortest path from the current cell of the agent to a cell with the smallest f-value among all cells for which the h-values have not changed. If such cells do not exist, aLSS-LRTA* and aRTAA* find a shortest path from the current cell of the agent to a cell with the smallest f-value, like LSS-LRTA* and RTAA*. This is a simpler way of moving the agent to avoid heuristic depressions.

We compared aLSS-LRTA*, daLSS-LRTA*, aRTAA* and daRTAA* with the real-time search algorithms LSS-LRTA* and RTAA*. Our results, shown in Figure 1, indicate that daLSS-LRTA* and daRTAA* outperform LSS-LRTA* and RTAA*, respectively, by one order of magnitude when the time per planning episode is small. Details are given in (Hernández and Baier 2012).

Local Searches

LSS-LRTA* and RTAA* have a performance issue due to performing local searches, that is, repeated A* searches around the current cells of the agent. We describe RTBA* and TBAA*, two real-time heuristic search algorithms for goal-directed navigation in a priori completely or partially unknown grids that address this issue in the context of the game time model. The game time model partitions time into time intervals of a given length of time. During each time interval, the agent is allowed to search for the given length of time and then execute a single move (or pass on the move). Performance is measured by the number of time intervals before the agent reaches the goal cell. This performance measure is more realistic for video games than the number of moves before the agent reaches the goal cell since agents in video games are not allowed to execute moves at arbitrarily high speeds.

TBA* (Björnsson, Bulitko, and Sturtevant 2009) is a real-time heuristic search algorithm for goal-directed navigation on a priori known grids that performs one global search, that is, an A* search around the start cell. It performs an A* search from the start cell to the goal cell until the goal cell is about to be expanded or the open list becomes empty. If the open list becomes empty, the agent stops unsuccessfully. At the end of each time interval, the agent makes one move along a path from its current cell to the cell with the smallest f-value found by the A* search, by either following the shortest path from the start cell to a cell with the smallest f-value (if its current cell is on this path) or by moving to the parent of its current cell in the A* search tree. If it reaches the goal cell, it stops successfully. TBA* often outperforms LSS-LRTA* and RTAA* since a global search increases the chances that the agent follows a short path from the start cell to the goal cell (Hernández et al. 2012). We describe

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves
0.3	3,245	3,244	2,879	2,878	4,613	4,604	2,290	2,286	7,155	2,004	3,230	2,010	**2,203**	2,027
0.6	2,598	2,597	2,472	2,471	3,368	3,360	2,147	2,144	4,487	2,004	2,572	2,010	**2,090**	2,027
0.9	2,451	2,450	2,418	2,417	2,918	2,910	2,101	2,099	3,611	2,004	2,361	2,010	**2,062**	2,027
1.2	2,310	2,309	2,305	2,304	2,695	2,688	2,086	2,083	3,178	2,004	2,260	2,010	**2,051**	2,027
1.5	2,281	2,280	2,272	2,271	2,560	2,553	2,070	2,068	2,920	2,004	2,202	2,010	**2,045**	2,027

Table 1: Evaluation of RTBA* and TBAA* on A Priori Completely Unknown Grids

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves	# Time Intervals	# Moves
0.3	2,694	2,693	2,460	2,459	2,734	2,730	**1,505**	1,504	6,324	1,409	2,430	1,399	1,659	1,418
0.6	2,039	2,038	1,863	1,862	2,037	2,034	**1,442**	1,441	3,812	1,409	1,875	1,399	1,532	1,418
0.9	1,840	1,839	1,779	1,778	1,860	1,857	**1,431**	1,430	2,979	1,409	1,695	1,399	1,490	1,418
1.2	1,707	1,706	1,643	1,642	1,726	1,724	**1,421**	1,420	2,564	1,409	1,608	1,399	1,470	1,418
1.5	1,620	1,619	1,642	1,641	1,668	1,666	**1,415**	1,414	2,316	1,409	1,556	1,399	1,458	1,418

Table 2: Evaluation of RTBA* and TBAA* on A Priori Partially Unknown Grids

two variants of TBA* for goal-directed navigation on a priori completely or partially unknown grids:

- RTBA* (Hernández et al. 2012) is almost identical to TBA*, the only difference is that, if the agent observes a blocked cell on the path from its current cell to the cell with the smallest f-value, RTBA* starts a new A* search from the current cell of the agent to the goal cell.
- TBAA* (Hernández et al. 2012) is almost identical to RTBA*, the only difference is that, like RTAA*, it sets the h-values of all states in the closed list to the largest possible h-values that maintain the consistency of all h-values before it starts a new A* search. To be precise, it actually defers an h-value update until the time when the h-value is needed during a future A* search to avoid computing those h-values that are not needed later. The h-value updates make the h-values more informed and thus focus future A* searches better.

We compared RTBA* and TBAA* with the real-time heuristic search algorithms RTAA* and daRTAA* as well as the heuristic search algorithms Repeated A*, Adaptive A* and D* Lite (using a different experimental setup from the previous section). Repeated A* performs a complete A* search from the current cell of the agent to the goal cell until the goal cell is about to be expanded or the open list becomes empty. If the open list becomes empty, the agent stops unsuccessfully. The agent then moves along the shortest path from its current cell to the goal cell and remembers all blocked cells that it observes. If it reaches the goal cell, it stops successfully. If it observes a blocked cell on the path, it repeats the process. Incremental heuristic search algorithms, such as Adaptive A* (Koenig and Likhachev 2006a) and D* Lite (Koenig and Likhachev 2002), are almost identical to Repeated A*, the difference is that they speed up the A* searches by using their experience with prior A* searches to speed up future ones. Adaptive A* performs A* searches from the current cell of the agent to the goal cell, while D* Lite performs searches in the opposite direction. Our results, shown in Tables 1 and 2, indicate that TBAA* has a slight performance advantage over D* Lite in a priori partially unknown grids and vice versa in a priori completely unknown grids, although the differences might not be statistically significant. In both cases, TBAA* has the advantage over D* Lite that the agent starts to move right away. Details are given in (Hernández et al. 2012).

Objectives of the Demonstration

Our demonstration consists of a poster, videos and interactive simulations of real-time heuristic search algorithms. It has the following objectives:

1. Our demonstration provides a brief introduction to real-time heuristic search by describing LSS-LRTA* and RTAA*.
2. Our demonstration illustrates a performance issue of LSS-LRTA* and RTAA* due to depressions in the h-value surface. It describes aLSS-LRTA*, daLSS-LRTA*, two real-time heuristic search algorithms that address this issue, and summarizes their properties described in (Hernández and Baier 2012).
3. Our demonstration illustrates a performance issue of LSS-LRTA* and RTAA* due to performing repeated A* searches around the current cells of the agent. It describes real-time heuristic search algorithms that address this issue, namely RTBA* and TBAA*, and summarizes their properties described in (Hernández et al. 2012).

Acknowledgments

This material is based upon research supported by NSF (while Sven Koenig was serving at NSF). It is also based upon research supported by ARL/ARO under contract/grant number W911NF-08-1-0468 and ONR in form of a MURI under contract/grant number N00014-09-1-1031. Jorge Baier was partly funded by Fondecyt grant number

11110321. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. 2009. TBA*: Time-bounded A*. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 431–436.

Bulitko, V.; Björnsson, Y.; Sturtevant, N.; and Lawrence, R. 2011. *Real-time Heuristic Search for Game Pathfinding*. Applied Research in Artificial Intelligence for Computer Games. Springer.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2).

Hernández, C., and Baier, J. A. 2012. Avoiding and escaping depressions in real-time heuristic search. *Journal of Artificial Intelligence Research* 43:523–570.

Hernández, C.; Baier, J. A.; Uras, T.; and Koenig, S. 2012. Time-Bounded Adaptive A*. In *Proceedings of the 11th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, 525–532.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 476–483.

Koenig, S., and Likhachev, M. 2006a. A new principle for incremental heuristic search: Theoretical results. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS)*, 402–405.

Koenig, S., and Likhachev, M. 2006b. Real-Time Adaptive A*. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 281–288.

Koenig, S., and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems* 18(3):313–341.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147(1-2):253–279.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

Zelinsky, A. 1992. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation* 8(6):707–717.

Integrating Vehicle Routing and Motion Planning

Scott Kiesel, Ethan Burns, Christopher Wilt and Wheeler Ruml
Department of Computer Science
University of New Hampshire

Abstract[1]

There has been much interest in recent years in problems that combine high-level task planning with low-level motion planning. In this paper, we present a problem of this kind that arises in multi-vehicle routing. It tightly integrates task allocation and scheduling, who will do what when, with path planning, how each task will actually be performed. It extends classic vehicle routing in that the cost of executing a set of high-level tasks can vary significantly in time and cost according to the low-level paths selected. It extends classic motion planning in that each path must minimize cost while also respecting temporal constraints, including those imposed by the agent's other tasks and the tasks assigned to other agents. Furthermore, the planner is part of an interactive system and must operate within soft real-time constraints. We present an approach based on a combination of tabu search, linear programming, and heuristic search. We evaluate our system on representative problem instances and find that its performance meets the demanding requirements of the application. Our work demonstrates how integrating multiple diverse techniques can successfully solve challenging real-world planning problems that are beyond the reach of any single method.

1 Abstract reproduced from the article accepted in the main technical track at ICAPS 2012.

EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization

Javier Barreiro[*], Matthew Boyce[*], Minh Do[*], Jeremy Frank[†], Michael Iatauro[*], Tatiana Kichkaylo[‡], Paul Morris[†], James Ong[+], Emilio Remolina[+], Tristan Smith[*], David Smith[†]

[*]SGT Inc., NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035

[†] NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035

[+] Stottler Henke Associates, Inc., 951 Mariners Island Blvd., Suite 360, San Mateo, CA 94404

[‡] Decision Systems, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292

Abstract[1]

EUROPA is a class library and tool set for building and analyzing planners within a Constraint-based Temporal Planning paradigm. This paradigm has been successfully applied in a wide range of practical planning problems and has a legacy of success in NASA applications. EUROPA offers capabilities in 3 key areas of problem solving: (1) Representation; (2) Reasoning; and (3) Search. EUROPA is a means to integrate advanced planning, scheduling and constraint reasoning into an end-user application and is designed to be open and extendable to accommodate diverse and highly specialized problem solving techniques within a common design framework and around a common technology core. In this paper, we will outline the core capabilities of this open-source planning & scheduling framework. While EUROPA is the complete planning and scheduling software suite, we will pay special attention to the aspects that are relevant to knowledge engineering: modeling support, embedding a planner into an end-user application, and plan visualization and analysis.

1 Abstract reproduced from the paper submitted to ICKEPS.

On Computing Conformant Plans Using Classical Planners: A Generate-And-Complete Approach

Khoi Nguyen, Vien Tran, Tran Cao Son, Enrico Pontelli
Computer Science Department
New Mexico State University, Las Cruces, NM 88003

Abstract[1]

Generate-and-complete is an approach to conformant planning that generates a possible plan for one possible world and completes the conformant plan by inserting more actions into the possible plan. They key idea is to exploit the interaction of actions in a plan with different possible worlds. The completion method is based on maintaining executability and effects of actions in different possible worlds while achieving goals. The approach also employs the one-of technique to reduce the number of possible worlds. We develop a system, called GC[LAMA], based on this approach and a classical planner, LAMA 2008. GC[LAMA] shows excellent coverage and performance compared to state-of-the-art conformant planners. The results verify that generate-and-complete is a very strong alternative to belief state space search approach.

1 This demonstration corresponds to a paper presented in the main technical track at ICAPS 2012.