ICAPS 2010

20th International Conference on Automated Planning and Scheduling

Proceedings of the System Demonstrations

Toronto, Canada – 14 May 2010

Edited by
Ivan Serina and Neil Yorke-Smith

The 2010 ICAPS Conference is sponsored by

Artificial Intelligence Journal, Elsevier

David E. Smith, USA

Australia's ICT Research Centre of Excellence (NICTA), Australia

Engineering and Physical Sciences Research Council (EPSRC), UK

National Science Foundation, USA

Held in cooperation with the Association for the Advancement of Artificial Intelligence

ICAPS 2010

Proceedings of the System Demonstrations

Edited by
Ivan Serina and Neil Yorke-Smith

The ICAPS 2010 logo was designed by Dafna Brafman

Cover painting courtesy Ingo Meironke

Contents

Papers

Preface

The Demonstrations and Exhibits programme at ICAPS 2010 provides an opportunity for planning and scheduling researchers and practitioners to demonstrate their state-of-the-art implementations in action. The sessions allow the community to experience the latest contributions while broadening the reach of novel methods in a relaxed social setting.

The 2010 Demonstrations and Exhibits programme follows the established antecedents at ICAPS. An innovation this year was the co-location of the main demonstration session with the AAMAS 2010 conference. The two conferences held an extended joint demonstration session on Friday 14 May. During this morning session, participants from both conferences interacted with exhibits spanning robotics, affective agents, scheduling, mixed-initiative planning, and multi-agent planning and coordination. ICAPS exhibitors subsequently showcased their work during the conference poster session. Votes from conference attendees were collected and an award for Best Demonstration was announced at the conference banquet.

The Demonstrations and Exhibits programme features ten systems ranging from deployed systems to research prototypes. Four of these systems were described in papers accepted to the main ICAPS conference.

Steve Chien and his colleagues at the NASA Jet Propulsion Laboratory and Goddard Space Flight Center demonstrate a timeline-based, heuristic greedy scheduling system in use to schedule observations for the Earth Observing One satellite. They describe the range of constraints modelled within the system and show a visualization of the scheduling search process with direct comparison to the prior scheduling system.

Florent Devin and Yannick Le Nir of the L@RIS - EISTI Laboratoire describe an approach to timetabling based on web services. They use a computational web service written in Prolog for handling the resources and the constraints, and an internet Application written in ZK, an open source AJAX web application framework. The application is integrated with Google Calendar, which is used to insert constraints and to view the final timetable.

Stefan Edelkamp and his colleagues of the University of Bremen and of the University of Dortmund examine how to translate concurrent C/C++ code into PDDL and propose a system that runs heuristic search planners against the PDDL outcome to generate traces for locating programming bugs. They demonstrate the aspects of parsing, generation of the dependency graph, slicing, abstraction, and property conversion.

Tara Estlin and her colleagues at the NASA Jet Propulsion Laboratory demonstrate a comprehensive multi-agent event detection, communication, and planning & scheduling system to enable coordination of multiple spacecraft assets for joint science campaigns. They show video footage, demonstration data, and software traces from a series of demonstrations completed in the JPL Marsyard, involving in-situ seismographic stations (landers), a rover, and two simulated spacecraft, and using communications infrastructure developed for the interplanetary internet.

Mark E. Giuliano of the Space Telescope Science Institute and Mark D. Johnston of the NASA Jet Propulsion Laboratory present tools that allow end users to effectively explore a set of solutions produced by multi-objective algorithms in order to select a single solution for execution. They present features of the Multi-User Scheduling Environment (MUSE) that provides the ability to visualize higher dimension objective value spaces, and for multiple users to converge on mutually acceptable schedules.

Malte Helmert of the Albert-Ludwigs-Universität Freiburg and Hauke Lasinger of LemnaTec GmbH introduce the Scanalyzer planning domain, a domain for classical planning which models the problem of automatic greenhouse logistic management. This domain was used as a benchmark in the sequential planning tracks of the last International Planning Competition. The competition results show that domain-independent automated planners can find solutions of comparable quality to those generated by specialized algorithms developed by domain experts, while being considerably more flexible.

Rajiv T. Maheswaran and his colleagues of the University of Southern California present a decision support system to help people coordinate in the real world when performing a shared task. The coordination is difficult since it requires simultaneously addressing planning, scheduling, uncertainty, and distributed locations. In the current approach, end-users can encode their intuition as guidance enabling the system to decompose large distributed problems into simpler problems that can be solved by traditional centralized AI techniques. The work comes from fielded trials of a large-scale project.

John Maraist and Robert P. Goldman of Smart Information Flow Technologies Inc. demonstrate a plan execution engine, called Shopper, that interprets the LTML plan language. LTML is an extension of the PDDL language with more expressive control structures and with support for semantic web services marked up in OWL-S. Goldman and Maraist show how one may program simulated web services (or other actions) for use in experimentation, and how Shopper operates with this simulation.

Javier Ortiz and his colleagues of the Universidad Carlos III de Madrid and Ericsson Research Spain demonstrate a data mining tool, called PDM (Planning for Data Mining), based on automated planning, that helps users to perform data mining tasks. Plans are data-mining knowledge flows, i.e., sequences of data mining actions that should be executed over the initial datasets to obtain the final models. Ortiz and colleagues demonstrate how any state-of-the art planner can be used to generate a plan (the current implementation uses SAYPHI), which is executed over the initial dataset to obtain the final model. Each plan or knowledge flow is executed by the machine learning engine WEKA.

Tran Cao Son and his colleagues of New Mexico State University examine three conformant planners (called CPA, DNF, and CNF) and compare them with other state-of-the-art conformant planners. CPA was the recipient of the Best Non-Observable Non-Deterministic Planner Award, IPC-2008. Their demonstration proposes techniques that contribute to the performance and scalability of these planners and exposes the lessons learned during the development of these planners.

These demonstrations illustrate the many-faceted nature of planning and scheduling applications and their underlying support technologies. The demonstrations present applications that cross scopes from spacecraft operations, web services, multi-agent planning and scheduling, and data mining tools. The goals of the applications range widely from reducing costs, to better utilizing resources, to planning in uncertain environment, to coordinating groups of users. Finally, the demonstrations illustrate the challenges involved in transferring applications from research laboratories to the real world.

This proceedings of the 2010 Demonstrations and Exhibits programme contains abstracts and extended abstracts that describe the systems showcased. Systems described in papers presented in the main ICAPS conference are summarized here with abstracts; we refer to the conference proceedings for their full description. In addition to this proceedings, supplementary materials such as videos are found on the ICAPS'10 website.

We would like to thank the participants of the Demonstrations and Exhibits programme, and all those that helped make the programme a success. We are grateful to the ICAPS and AAMAS Chairs Joerg Hoffmann, Henry Kautz, Michael Luck and Sandip Sen, and to the AAMAS Demonstrations Chairs Catherine Pelachaud and Iyad Rahwan. We express our gratitude to the Local Arrangement Committees, chaired by Jorge Baier and Yves Lesperance, and to the AAAI staff, for their generous efforts in arranging and organizing the events in Toronto.

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

– Ivan Serina and Neil Yorke-Smith
Exhibits and Demonstrations Co-chairs

Organizing Committee

Ivan Serina
Free University of Bozen-Bolzano, Italy

Neil Yorke-Smith
American University of Beirut, Lebanon and SRI International, USA

A Demonstration of Timeline-based Scheduling for the Earth Observing One Mission

Steve Chien and **Daniel Tran** and **Gregg Rabideau** and **Steve Schaffer**
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
Firstname.Lastname@jpl.nasa.gov

Daniel Mandl and **Stuart Frye**
NASA Goddard Space Flight Center
Greenbelt, MD 20771, USA
{daniel.j.mandl, stuart.w.frye}@nasa.gov

Abstract

We demonstrate a timeline-based heuristic greedy scheduling system in use to schedule observations for the Earth Observing One Satellite. We describe the range of constraints modeled within the system directly and as part of the candidate generation process. We show a visualization of the scheduling search process with direct comparison to both the prior scheduling system and simplified optimal upper bound schedulers. We present results documenting that our heuristic scheduler produces results within 15% of the optimal upper bound and a significant (50%) increase in scenes over the prior scheduler with an estimated value of missions of dollars US.

Timetabling RIA in action

Florent Devin and **Yannick Le Nir**
L@RIS - EISTI
26 avenue des lilas
64062 Pau Cedex 9
FRANCE

Abstract

We present an Rich Internet Application to create timetabling for real engineering school. In this application, we have resources and constraints. Resources describe the timetable's context. Constraints are restrictions of resources. All lectures are placed on a timetable composed of time-slots. For timetabling, we use a computational web service written in Prolog. Our application presents two different views, one for the user, and one for the *admin*.

Users can view all generated timetables and put in their own constraints. The input of constraints can be done by two ways, directly from our application, or via Google calendar. To use Google calendar, users have to update the link of their Google calendar.

Admin has many tasks to do. He has to plan all different lectures that occur in a year. He also has to specify which contributors teach which lecture, and to whom the lecture is being given. Moreover, he has to input constraints for classes, and possibly rooms' unavailability. Then he has to validate or invalidate users constraints. Finally he can generate one or more timetables. He can also export all generated timetables to Google calendar.

Introduction

In this paper we present an original approach to timetabling. This approach is based on the concept of web services. To be able to create timetables, there are two different parts in our application. One is the Rich Internet Application (RIA), and the other is the computational part. Before viewing how our application runs, we have to define several terms. Then we explain why we have chosen to create an RIA. Finally, we describe the demonstration itself.

Definitions

First we have to define *time-slot* and *timetable*. For us a time-slot is a period with a start time and a fixed duration. So a time slot is the minimal time interval we can find on a timetable. A timetable is a consecutive list of time-slots on which resources are planned.

Then we have to define *resources* describing the context of our application. In fact, we consider three resources:

- main resources: elements to plan (lectures, meetings, . . .);
- static resources: elements which are linked to main resources (contributors, classes, . . .);
- dynamic resources: elements to include during the computation (rooms, materials, . . .).

Finally we have to define *availability* of resources. This is all the possible associations between resources and the amount of consecutive time-slots. *Unavailability* is the complementary of *availability* in the timetable. Unavailability can also called a *constraint*.

Rich Internet Application

Rich Internet Application provides a very viable technology (Duhl 2003), offering most desktop features. This technology can address many users, without any requirement, because they are run on a web browser. By using a web browser, users do not have to learned either a particular operating system, or a particular software to use our application[1] (Rogowski 2007), (Driver and Rogowski 2007).

Our application is written with the ZK framework. ZK is an open source Ajax web application framework (Seiler 2009). Yeh(2006) shows numerous advantages of using this framework. Also ZK uses a centric server approach, which simplifies the security of the application, and saves time during the coding phase. Thus our application is written with Java, and we can use Hibernates' tools.

CSP and Prolog

Constraint Satisfaction Problem can efficiently model the timetabling problem (Wallace 1996), (Frühwirth and Abdennadher 2003), (Jaffar and Maher 1994), (Carter and Laporte 1998). We can translate the timetabling problem with a CSP on finite domain. The implementation of the computation is written with Prolog. It communicates with the RIA using web services, and also directly with our database using a classic ODBC.

Application's architecture

Before describing our demonstration, we now introduce the general architecture of our application. The central point, for users, is the RIA. This RIA communicates with other

[1] Except our application

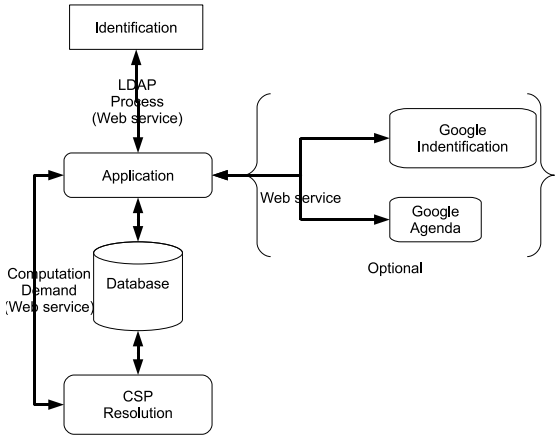

Figure 1: General architecture system

applications using web services. Currently, we have three web services:

- Identification process: as our application is used by our contributors, we have to include it in the existing IT system. When someone tries to log into our application, we ask the LDAP[2] system if the user is authorizes to access our IT system, via a web service.
- Computation process: the computation process is written in Prolog. The RIA is written in Java. In order to communicate with each other, RIA and Prolog have to interact. For the communication from the RIA to the Prolog part, we use a web service.
- Constraints process: we can use Google calendar to input constraints as the demonstration will show. Using Google services is like using web services.

To store data, we use a database. This database is a MySQL database. The RIA create/retrieve/update or delete data with the help of Hibernate. The Prolog part, as mentioned above, uses a classical ODBC.

Timetabling RIA in action

Our demonstration will show the two different views of the application, the one for users, and the one for the admin. In this demonstration, you will see a lot of timetables. You will note that the timetable has many colored time-slots. In fact, there is one color for one feature's classroom.

User demonstration

User can view all previously generated timetables, as timetables are public[3] information. There are three different views for timetables, one for contributor, one for class, and one for classroom. A user can also modify his personal data, like a link to his own Google calendar. He can also put unavailability. A screen is dedicated to this function. In this screen, he can see all previously input constraints. He can also view the state of the constraint, that is to say if the constraint has been validated, or invalidated by the admin. User can do nothing more.

Admin demonstration

The most important part of our demonstration is for the admin's functions. The admin have many tasks to do:

- generation of timetables;
- constraint keyboarding/checking;
- lectures planning function;
- lectures adjustment;
- contributors lectures association.

At the very beginning we have to create the classes, and moreover the subgroups in each class. For example, as you can see in the video, the class named *"ING 1"* is divided into two subgroups, named *A* and *B*. Doing this, we also specify the size of each group, this will be useful later when we want to create the timetable to consider a room's size and group's size.

We may, also at the very beginning, define all existing classroom. A classroom is defined by its location, features and size. This step is essential as the computation considers room's size and feature to plan a course.

Then we have to specify which lectures can occur for which class. Which means that we have to name the lecture, and associate the class with it. Over the years this step does not have to be repeated, as lectures for a year do not often change.

Then we have to plan in advance the lectures that occur during a year, a term, or a period. This phase is shown on the third admin's video. By planning the lecture in advance, we also specify the number of theoretical courses per week, and also the number of practical courses per week. At the same time, we also indicate the duration of courses (per week). The difference between theoretical courses and practical courses is that theoretical courses are done for the entire class once, and practical courses are done once for each subclass.

Once this is done, we can associate the contributors, as shown also on the third video. We do not create contributors, as we can synchronize the contributors via our LDAP services. The only thing to do is to synchronize our database. Once our database is updated, we can indicate which contributor teach which courses, and also what is the feature they need.

With this data, we can generate a valid timetable, if contributor does not have any constraints. To generate a timetable, we just have to choose the week, or a starting week and an ending week, and ask for computation. This can take time, about 20 seconds for a week. And you will note that using a web service is totally appropriate. During this step we can also ask for the export to Google calendar. If we ask for it, all the generated timetables will be exported. The export is done for all contributors, classes, classrooms calendar. If contributors have their own calendar, we have control the other calendars. These can be shared to offer

[2]LDAP stands for Lightweight Directory Access Protocol

[3]By public, we mean that all contributors can be informed of other timetables.

a complete view of the timetable for students, contributors, and administration.

There is also a screen for changing a lecture in duration for a particular week. We can also change the number of courses for a week. For instance, if we have planned a lecture occurring once a week both for theoretical and practice, we can assume that for a particular week, there is no practical course, but the theoretical course is longer. If we are in this case, or whatever similar case, we will use this screen.

We can also deal with constraints. To do this, there are two screens, one more textual (as is shown on the user's video), and one more graphical (as is shown on the second admin's video).

As shown on the first admin's video, we can put in an exceptional event. This event will be considered by the computational part as a hard constraint.

All this functionality, user and admin, will be shown during the demonstration, except the configuration phase. The configuration phase is the keyboarding of contributors, the naming of lectures, and the specification of rooms and classes. If the audience wants, we can show this phase too.

Conclusion

We present an RIA for timetabling, which is fully functional. This RIA uses web services to timetabling. It is important to note that the workload of the admin is considerably reduced by using our application rather than handwriting timetables. We have paid a particular attention to simplifying the admin's works. Also this application can evolve by the use of other web services. We also consider users' habits, by providing a way to use their own calendar.

References

Abbas, A., and Tsang, E. 2001. Constraint-based timetabling-a case study. *Computer systems and applications, ACS/IEEE international conference on* 0:0067.

Abdennadher, S.; Aly, M.; and Edward, M. 2007. Constraint-based timetabling system for the german university in cairo. In *INAP/WLP*, 69–81.

Carter, M. W., and Laporte, G. 1998. Recent developments in practical course timetabling. In *PATAT '97: Selected papers from the second international conference on practice and theory of automated timetabling II*, 3–19. London, UK: Springer-Verlag.

Driver, E., and Rogowski, R. 2007. Rias bring people-centered design to information workplaces. Forester Research.

Duhl, J. 2003. White paper : rich internet application. IDC.

Frühwirth, T., and Abdennadher, S. 2003. *Essentials of constraint programming*. Springer Verlag.

Jaffar, J., and Maher, M. J. 1994. Constraint logic programming: a survey. *J. Log. Program.* 19/20:503–581.

Qu, R.; Burke, E. K.; Mccollum, B.; Merlot, L.; and Lee, S. Y. 2009. A survey of search methodologies and automated system development for examination timetabling. *J. of scheduling* 12:55–89.

Rogowski, R. 2007. The business case for rich internet application. Forrester Research.

Seiler, D. 2009. Ria with zk. In *JAZOON09*.

Wallace, M. 1996. Practical applications of constraint programming. *Constraints* 1:139–168.

Yeh, T. M. 2006. Zk ajax but non javascript.

Automated Program Checking via Action Planning

Stefan Edelkamp
University of Bremen
Germany

Mark Kellershoff
Dortmund University of Technology
Germany

Damian Sulewski
University of Bremen
Germany

Abstract

In this paper we translate concurrent C/C++ code into PDDL. The system then runs heuristic search planners against the PDDL outcome to generate traces for locating programming bugs. These counter-examples result in an interactive debugging aid and exploit efficient planner in-built heuristics. Different aspects like parsing, generation of the dependency graph, slicing, abstraction, and property conversion are described. For data abstraction we provide a library, and for increased usability the tool has been integrated in Eclipse.

Introduction

Planning via model checking (Cimatti et al. 1997) considers the integration of verification technology into AI planners. For model checking via planning by considering the rising effectiveness of planning search heuristics in verification (Wehrle and Helmert 2009), a natural question is to apply planning technology directly. The effectiveness of translating model checking inputs into PDDL has been documented by a series of preceding papers, including the *communication protocols* (Edelkamp 2003), *Petri nets* (Edelkamp and Jabbar 2006), and *μ-calculus formulae* (Bakera et al. 2008), and *graph transition systems* (Edelkamp, Jabbar, and Lluch-Lafuente 2005).

In contrast to *model checking* (Clarke, Grumberg, and Peled 1999) that relies on a model of the system to be checked, *program checking* (Visser et al. 2003) aims at the automated verification of programs given its source code by analyzing the compiled executable. Typically, tools operate on top of a virtual machine that has been extended to simulate different execution branches. Verification units that consider checking the object code like *JPF* (Visser et al. 2000) *Steam* (Leven, Mehler, and Edelkamp 2004) and *MoonWalker* (de Brugh, Nguyen, and Ruys 2009), complement bounded software model checking tools like *CBMC* (Clarke, Kroening, and Lerda 2004).

This paper proposes the automated transformation of C/C++ sources into PDDL (Fox and Long 2003) to exploit refined guidance inherent to heuristic search planners. The rationale of applying heuristics is that *directed* model checkers (Edelkamp et al. 2008) quickly report short counter-examples. The rationale for a PDDL encoding are accurate planning heuristics (Helmert and Domshlak 2009).

The choice of the C/C++ is urged by its wide-spread use, rising thread parallelism in programs for the support of multi-core machines, and the lack of advanced bug-finding support. Tools like *valgrind* are able to find memory leaks, but not to validate concurrent programs. As we concentrate on the imperative core of C/C++, the results likely generalize to other programming languages like Java or Ruby.

The transformation into Level 2 PDDL is able to directly uncover bugs. For the case a program cannot be analyzed completely, different abstractions apply. Besides *slicing* the program without loss of information, *data abstraction* converts infinite state variables to finite domain, and to Level 1 PDDL. Dependencies among variables are automatically detected by analyzing the parse of the source.

Annotated Parse and Dependency Graph

In program checking, sources are analyzed that have non-deterministic effects. Such non-determinism can be due to the interleaving of concurrent threads, unknown assignments to variables, program and user inputs, explicit choice points imposed by the programmer, or abstractions of deterministic programs.

As C/C++ is a rather complex language (Stroustrup 1994), we adapted JavaCC by Sreenivasa Viswanadha (published in 1997) to parse the input. The parser yields an abstract syntax tree, which we present as a navigational aid to the programmer, and which is used for further processing. For verification, the C/C++ code is annotated with a small set of commands for its controlled execution:

- `VLOCK(<variable>)` restricts the access to the variable `<variable>` in the currently invoked thread.
- `VUNLOCK(<variable>)` releases the lock to the variable `<variable>`.
- `BEGINATOMIC()` dictates that the current thread cannot be suspended.
- `ENDATOMIC()` terminates the atomic block selection within a thread.
- `VASSERT(<condition>)` tracks `<condition>` to be satisfied each time the program reaches it.
- `RANGE(<variable>,<low>,<high>)` offers non-deterministic choices to a program.

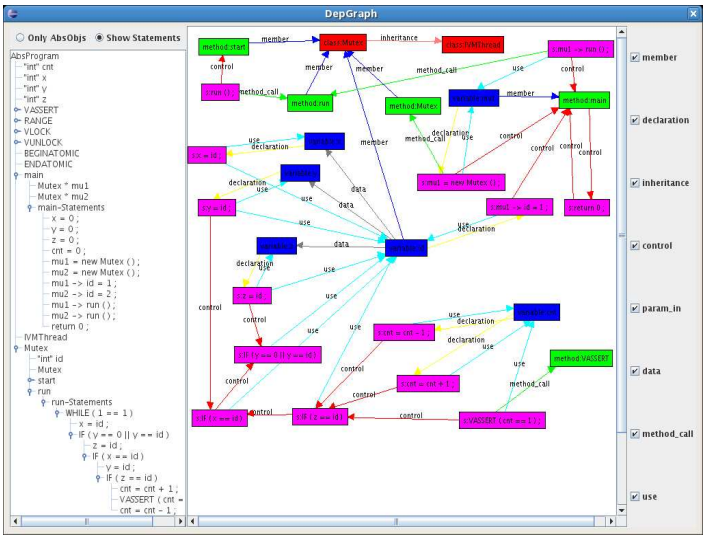

Figure 1: Dependency graph of a C/C++ program.

For transforming the source code, the input has to be made avaailable in a dictionary data structure, while supporting the conversion from infix (as used in C/C++) to prefix notation (as used in PDDL). We implemented a hierarchy of containers. Its structure represents the scopes of a program and is exploited for constructing the object-oriented *dependency graph*. An example is shown in Fig.1 together with parts of the parse tree to its left.

The state of a C/C++ program includes information like assignments to global and local variables, as well as stack and dynamic memory contents. We assume variables of type boolean, integer and real. The situation before the execution of a program is called initial state, and the state of a program at its termination is called (valid) end state. Additionally to the variable assignments, a state contains information about the program counter, denoting which transition has been or will be executed. If the program is multi-threaded, a program counter is maintained for every running thread including the thread for *main*. For the conversion, we assume that a static analysis, applied after parsing the code, can detect the number of threads running concurrently.

The Translation into PDDL

The core motivation of translating the source of a program into PDDL is to use planner in-built heuristic to drive the exploration process towards falsifying a property, e.g. in form of a deadlock, a failed assertion/global invariance, or an array access violation.

The output pleases the first two levels from the PDDL hierarchy (Fox and Long 2003). In PDDL Level 1, states are collections of true facts. It allows quantification over domain objects, disjunctive and negated preconditions, as well as conditional effects. In PDDL Level 2, real-valued fluents are available. Preconditions of actions cover arithmetic expressions over the variables, while the effects can additionally modify value assignments.

Conversion of Variables For a C/C++ variable *declaration*, like *int a*, we reserve a PDDL variable *int_a* (allowing real value assignments). Since a program can contain several variables with name a, every PDDL variable is suffixed with an additional id, such that for our case we infer *int_a_1*, as it is the first (and only) appearance of a that is converted. As a can appear in different threads, we provide an additional parameter to the PDDL predicate, yielding the expression *(int_a_1 ?t - thread)* to represent the variable declaration of a. For variable *int b* the conversion is analogous. In short terms, variable conversion is a mapping that assigns a planning variable to each program variable.

Variable Assignments For translating an *assignment* to a variable into PDDL, we construct actions, which convert the state in the planning model in the same way it does within the program. As we have the parse of the expression available, the conversion from in- to prefix notation is immediate. The parameter is the thread, while the effect changes the planning state equivalent to the assignment *a=1000;*:

```
(:action SimpleAssignment
:parameters (?t - thread)
:precondition (<predecessor has finished>)
:effect (and (assign (int_a_1 ?t) 1000 )
     (<this action has finished>)
     (not (<predecessor has finished>))))
```

Control Flow We use predicates to model line numbers. Every action includes as a precondition that the predecessor (line) has finished its execution. For example, in a sequential execution line 20 has to finish before line 21 is processed. To avoid ambiguities, every line number is attached to the file in which the line is contained. It is also parameterized with the thread that is invoked. Since in PDDL every possible action is checked for execution, preconditions have been enlarged to select the actions that are currently activated.

The action for the first line in the main program includes *(start T0)* as a precondition triggered by the initial state, since it does not have a direct predecessor. The first instruction of a method also contains a label that it has been called.

Consider the following simple example program fragment

```
#include "Thread.h"
[...]
class Example:public Thread {
public:
 Example();
 void run();
};
Example::Example(){}
  void Example::run(){
    int a;
    int b;
    a=1000;
    b=20;
    VASSERT(a < b);
 }
```

After calling *run*, we have two concurrent threads: *main* (thread t_0), and the thread (t_1) that has been called. If we omit the details for thread invocation, the remaining program logic has to include the variables a and b, their order

and the constraints imposed. Hence, the PDDL equivalent for the assignments *a=1000* and *b=20* is as follows.

```
(:action Example_cpp_Line_20
:parameters (?t - thread)
:precondition (Example_cpp_Line_19 ?t)
:effect (and (assign (int_a_1 ?t) 1000 )
    (Example_cpp_Line_20 ?t)
    (not (Example_cpp_Line_19 ?t))))
(:action Example_cpp_Line_21
:parameters (?t - thread)
:precondition (Example_cpp_Line_20 ?t)
:effect (and (assign (int_b_2 ?t) 20)
    (Example_cpp_Line_21 ?t)
    (not (Example_cpp_Line_20 ?t))))
```

An *if-statement* exists in two different variants (with or without else block). A model without an else-branch is not directly convertible in PDDL, as failing the if-condition would not increase the program counter. We observe that every action has access to its immediate predecessor, and every action has at most two successors in case of a branch and two predecessors in case of a join. A *while-statement* is an if-statement featuring a backward jump.

For if-statements we introduce a *virtual-else* branch. The if-statement itself would vanish as the conditions are imposed as additional preconditions to the actions. But without modeling the if-statements explicitly, there is a subtle problem in modeling nested if-statements. Consider the small extension of the running example in Fig. 2. If one uses one action per instruction, then implementing correct precedences among the if statements is tricky, i.e., to connect an else-branch to the corresponding if. Therefore, we decided to include an additional flag and an additional action for starting and ending an if- or else- part.

```
void Example::run(){
  int a;
  int b;
  a=1000;
  b=20;
  if (a > b) {
    if (a > 20) {
       a=20;
    } else {
       a=0;
    }
  } else {
    a=1000;
  }
  VASSERT(a < b);
}
```

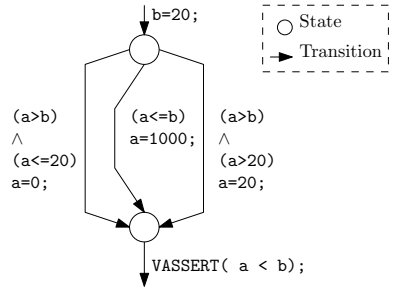

Figure 2: A nested if-statement and induced control flow.

Fig. 3 relates the source of a simple while-statement to the according automaton, that is used to monitor the flow of control in the PDDL code.

Model Checking Statements An *assert-statement* is split into two parts. One branch considers the violation of the assertion, in which case the predicate *assertionviolation* is set, the other branch continues with the flow of instructions.

```
void Example::run(){
  int a;
  int b;

  a=1000;
  b=20;
  while (a > b) {
      a=20;
      b=1000;
  }
  VASSERT(a < b);
}
```

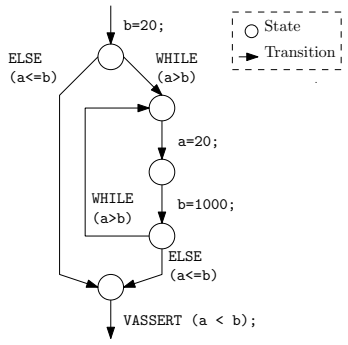

Figure 3: A while program and induced control flow.

When searching for the violation of safety properties, this predicate is included as a goal condition.

The *lock-statement* denotes that a thread requires exclusive access to a variable. The first thread that locks the variable has top priority, such that all upcoming accesses to the same variable are rejected. The PDDL model is extended in the sense that the actions include a precondition for locked variables, while avoiding multiple locks. A proper locking mechanism yields checks of invalid end states. A supplementary action wait is generated, that indicates that a thread waits for a resource. If all threads are blocked, a deadlock has occurred, an a goal achiever is triggered.

For *atomic blocks*, in the PDDL model 2 new predicates are inserted: *atomic* denoting that the execution is in atomic mode, and *isatomic ?t - thread* denoting which thread is actually atomic. Most ordinary actions are extended by the following precondition $atomic \Rightarrow (isatomic\ ?t)$. The end of the block generates an action without further specialized preconditions that deletes *atomic* and (*isatomic ?t)*.

Complex Statements For *indexed variable access* first the index is determined then the access to the array is executed. As PDDL does not provide a mechanism to index variables with numbers, we allow the user to adjust upper bounds provided by the parser (in PDDL 3.1, indirect variable access is available but only a few planners support the extension). The conversion of *C++-objects* into PDDL is possible, if the initialization uses the new-operator and gets assigned to a unique name. The *new-statement* induces the reservation of a PDDL object with a reference to this object; the variables of the class contain an additional parameter, whose type is the class name.

Methods PDDL models cannot generate objects dynamically. The only *methods* that are currently supported are those that have $integer^* \rightarrow integer$ or $integer^* \rightarrow void$ in their signature.

Methods are converted in actions that are triggered by setting a special predicate. The parameters are found on the method-stack, and the solutions are found in a special solution register, accessible from the calling action, similar to what is done in an ordinary executable. Actions are indexed s.t. more than one call is possible.

	Mutex			Producer-Consumer			BubbleSort10			8-Puzzle		
Program	l	s	t	t	s	t	l	s	t	l	s	t
StEAM DFS	66	742	90	122	353	59	594	1184	124	36096	70366	6552
StEAM BF	30	1630	198	29	4383	690	594	1774	88	86	7836	269
FF EHC	27	747	6	20	23	2	-	-	-	-	-	-
FF Best-First	27	251	4	20	3405	10	-	-	-	-	-	-
FF DA EHC	28	39	28	20	33	2	661	670	1312	115	932	2585
MFF DA EHC	28	39	108	20	34	56	661	670	8938	361	56257	1177572
MFF DA BF	27	922	877	20	3405	1609	661	73294	79674	99	4113	96291

Figure 4: Results for benchmarks; l denotes the length of the counter-example, s the number of states, t the CPU time in milliseconds, DA data abstraction, BF best-first search, and EHC enforced hill climbing applied in FF/MFF (MetricFF).

Abstraction We mainly support *data abstraction* (Merino et al. 2002). In the *neg-pos-zero* abstraction, for example, integers are projected to three values of being either positive negative, or equal to zero. If two negative or two positive values are multiplied, the result is determined, while for mixed multiplication, different options are possible. An alternative is an *odd-even* abstraction with obvious semantics.

Numerical abstractions have be implemented using abstraction libraries. The interface serves as a macro that is automatically extended to enrich the initial state and the PDDL operators to realize abstraction. The dependency graph then helps to deduce the set of all variables that are affected.

Programming Environment

The implementation used the following components: Eclipse 3.3 - Europa + CDT, the planner Metric-FF (Hoffmann 2003), and Java SDK 6 (includes Java-Script). The abstraction plugin that we have developed (see Fig. 5) consists of the GUI for parameterizing the algorithms, the parser, the dependency graph data structure, and the error trailer.

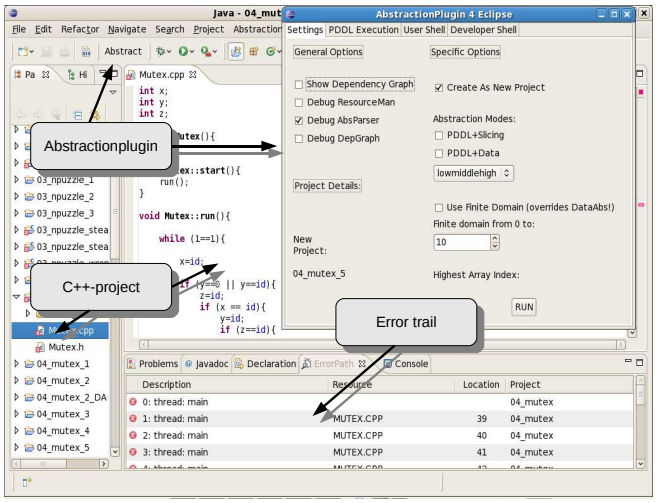

Figure 5: Abstraction Plugin and Error Trailer in Eclipse.

Figure 4 compares the performance for some simple C++ benchmarks with the one of the C/C++ program checker StEAM (Mehler 2006), which systematically analyzes a program as an executable in object code and complies with in- and output. In the *BubbleSort* and *8-Puzzle*, the program checker is faster, while in planning only data abstraction solves the problem. In the communication protocol examples, the analysis via PDDL is superior.

We do not claimed to have a full translation of C/C++ to PDDL. For example given that current PDDL is inherently static (it does not allow dynamic object creation), there are obvious restrictions to the expressiveness of sources that can be processed (no dynamic memeory allocation, no incremental invocation/deletion of threads etc). Nonetheless, the results indicate that PDDL can yield exploration advances.

References

Bakera, M.; Edelkamp, S.; Kissmann, P.; and Renner, C. D. 2008. Solving solving μ-calculus parity games by symbolic planning. In *MoChArt*, 15–33.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for AR. In *ECP*, 130–142.

Clarke, E. M.; Grumberg, O.; and Peled, D. A. 1999. *Model checking*. MIT Press.

Clarke, E. M.; Kroening, D.; and Lerda, F. 2004. A tool for checking ANSI-C programs. In *TACAS*, 168–176.

de Brugh, N. H. M. A.; Nguyen, V. Y.; and Ruys, T. C. 2009. Moonwalker: Verification of .net programs. In *TACAS*, 170–173.

Edelkamp, S., and Jabbar, S. 2006. Action planning for directed model checking of Petri nets. *Electronic Notes in Theoretical Computer Science (ENTCS)* 149(2):3–18.

Edelkamp, S.; Schuppan, V.; Bosnacki, D.; Wijs, A.; Fehnker, A.; and Aljazzar, H. 2008. Survey on directed model checking. In *MoChArt*, 65–89.

Edelkamp, S.; Jabbar, S.; and Lluch-Lafuente, A. 2005. Action planning for graph transition systems. In *ICAPS'05-Workshop on Verification and Validation of Model-Based Planning and Scheduling Systems*.

Edelkamp, S. 2003. Promela planning. In *SPIN*, 197–212.

Fox, M., and Long, D. 2003. PDDL2.1: An extension of pddl for expressing temporal planning domains. *JAIR* 20:61–124.

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

Hoffmann, J. 2003. The Metric-FF planning system: Translating *ignoring delete lists* to numeric state variables. *JAIR* 20:291–341.

Leven, P.; Mehler, T.; and Edelkamp, S. 2004. Directed error detection in c++ with the assembly-level model checker steam. In *SPIN*, 39–56.

Mehler, T. 2006. *Challenges and Applications of Assembly-Level Software Model Checking*. Ph.D. Dissertation, University of Dortmund.

Merino, P.; del Mar Gallardo, M.; Martinez, J.; and Pimentel, E. 2002. αSPIN: Extending SPIN with abstraction. In *SPIN*, 254–258.

Stroustrup, B. 1994. *The C++ Programming Language – 2nd ed.* Addison-Wesley Publishing Company.

Visser, W.; Havelund, K.; Brat, G.; and Park, S. 2000. Java pathfinder - second generation of a java model checker.

Visser, W.; Havelund, K.; Brat, G.; Park, S.; and Lerda, F. 2003. Model checking programs. *Automated Software Engineering Journal* 10(2):203–232.

Wehrle, M., and Helmert, M. 2009. The causal graph revisited for directed model checking. In *SAS*, 86–101.

A Demonstration of Multi-agent Event detection, Communications, and Planning & Scheduling to enable Coordinating Multiple Spacecraft Assets for Joint Science Campaigns

Tara Estlin, Steve Chien, Rebecca Castano, Daniel Gaines, Charles de Granville, Joshua Doubleday, Robert C. Anderson, Russell Knight, Benjamin Bornstein, Gregg Rabideau, and Benyang Tang

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109-8099, USA

We demonstrate the application of: multi-agent organization, automated science event detection [Castano et al. 2005], inter-agent communication via interplanetary internet [Burleigh et al. 2006], and automated planning & scheduling [Estlin et al. 2007, Chien et al. 2000] to enable opportunistic science observations to be autonomously coordinated between multiple spacecraft. Coordinated spacecraft can consist of multiple orbiters, landers, rovers, or other in-situ vehicles (such as an aerobot). Opportunistic science detections can be cued by any of these assets where additional spacecraft are requested to take further observations characterizing the identified event or surface feature (for a more complete description see [Estlin et al. 2010]).

We show video footage, demonstration data, and software traces from a series of demonstrations completed in the JPL Marsyard involving in-situ seismographic stations (landers), a rover [Schenker et al. 2001], and two simulated spacecraft (similar to [Chien et al. 2005] on EO-1), using communications infrastructure developed for the interplanetary internet [Burleigh et al. 2006]. The demonstration shows parts of detection and response for atmospheric events adapted from software operation on the MER rovers on Mars [Castano et al. 2007] as well as seismographic events [Huang et al. 2010], highlighting the ability of multiple assets to observe the same phenomena from multiple complementary perspectives (e.g., as in [Chien et al. 2005] for terrestrial applications).

Acknowledgement

This demonstrations utilized in-situ hardware packaged developed by the USGS Cascade Volcano Observatory, Mount Saint Helens (Richard Lahusen POC, Hardware and design) and node control software by the Washington State University (WenZhan Song and Behrooz Shirazi POCs node software).

References

A. Castano, A. Fukunaga, J. Biesiadecki, L. Neakrase, P. Whelley, R. Greeley, M. Lemmon, R. Castano, S. Chien, “Automatic detection of dust devils and clouds at Mars,” Machine Vision and Applications, 2007.

R. Castano, T. Estlin, R. C. Anderson, D. Gaines, A. Castano, B. Bornstein, C. Chouinard, M. Judd, “OASIS: Onboard Autonomous Science Investigation System for Opportunistic Rover Science,” Journal of Field Robotics, Vol 24, No. 5, May 2007.

S Chien, S., Sherwood, R., Tran, D., Cichy, B., Rabideau, G., Castano, R., Davies, A., Mandl, D., Frye, S., Trout, B., Shulman, S., and Boyer, D., "Autonomy Flight Software to Improve Science Return on Earth Observing One," Journal of Aerospace Computing, Information, and Communication, April 2010.

R. Huang, M. Xu, N. Peterson, W. Song, B. Shirazi, R. LaHusen, J. Pallister, D. Dzurisin, S. Moran, M. Lisowski, S. Kedar, S. Chien, F. Webb, A. Kiely, J. Doubleday, A. Davies, D. Pieri, “Optimized Autonomous Space In-situ

Sensor-Web for Volcano Monitoring," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 2010.

S. Chien, B. Cichy, A. Davies, D. Tran, G. Rabideau, R. Castano, R. Sherwood, D. Mandl, S. Frye, S. Shulman, J. Jones, S. Grosvenor , "An Autonomous Earth Observing Sensorweb," IEEE Intelligent Systems, May-Jun 2005, pp. 16-24.

T. Estlin, D. Gaines, C. Chouinard, R. Castano, B. Bornstein, M. Judd, and R.C. Anderson, "Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution," Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2007), Rome Italy, April 2007

R. Castano, T. Estlin, R. C. Anderson, D. Gaines, A. Castano, B. Bornstein, C. Chouinard, C., and M. Judd, "OASIS: Onboard Autonomous Science Investigation System for Opportunistic Rover Science," Journal of Field Robotics, Vol 24, No. 5, May 2007.

S. Burleigh, E. Jennings, E., and J. Schoolcraft, "Autonomous Congestion Control in Delay-Tolerant Networks", Proceedings of the Ninth International Conference on Space Operations, 19-23 June 2006, Rome, Italy.

NASA Successfully Tests First Deep Space Internet, NASA press release 08-298, 18 November 2008.

P. Schenker, E. Baumgartner, P. Backes, H. Aghazarian, L. Dorsky, J. Norris, T. Huntsberger, Y. Cheng, A. Trebi-Ollennu, M. Garrett, B. Kennedy, A. Ganino, R. Arvidson, S. Squyres, "FIDO: a Field Integrated Design & Operations rover for Mars surface exploration," Proceedings of the International Symposium on Artificial Intelligence, Robotics and Autonomous for Space, Montreal Canada, June, 2001.

S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, G., "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling, Breckenridge, CO, 2000.

Visualization Tools for Multi-Objective Scheduling Algorithms

Mark E. Giuliano and Mark D. Johnston

Space Telescope Science Institute
3700 San Martin Drive
Baltimore, MD 21218
Giuliano@stsci.edu

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive, Pasadena, CA 91109
mark.d.johnston@jpl.nasa.gov

Extended Abstract

Multi-objective algorithms for scheduling offer many advantages over the more conventional single objective approach. By keeping user objectives separate instead of combined, more information is available to the end user to make trade-offs between competing objectives. Unlike single objective algorithms, which produce a single solution, multi-objective algorithms produce a set of solutions, called a Pareto surface, where no solution is strictly dominated by another solution for all objectives. Algorithms for solving multi-objective scheduling problems have been developed that are effective in building a uniformly sampled approximation of the Pareto surface. The goal of this demonstration is to present tools that allow end users to explore the Pareto surface trade-off space in order to select a single solution for execution. This is challenging in at least two manners: First, the objective trade off space often has a high dimensionality making it hard for users to see patterns in the data using conventional graphical interfaces; Second, the nature of many multi-objective scheduling problems requires multiple users to be heavily involved, each such user contributing one or more objectives that reflect their interest in the outcome of the scheduling process. We present features of the Multi-User Scheduling Environment (MUSE) that provides the ability to visualize higher dimension objective value spaces, and for multiple users to converge on mutually acceptable schedules.

System Architecture

The visualization tools described below are part of the Multi-User Scheduling Environment (MUSE) overall architecture, as illustrated in Figure 1 (Johnston and Giuliano 2009, Johnston and Giuliano 2010). MUSE provides a generic environment for integrating existing tools (where they exist), providing persistent storage for various types of schedule data, and supporting both online and offline collaboration (in consideration of distributed users working across multiple time zones). MUSE incorporates server components (Fig. 1 lower half) as well as components that are resident on the user's workstation. MUSE distinguishes generic components (left) from those that may be highly domain specific (right). The architecture is designed so that domain specific components can be run as separate processes or can be compiled into the same image as the generic code.

On the server side, the Multi-Participant Coordinator acts as a central "clearing house" for schedule data, participant's selections, and scheduling runs. It provides an interface that communicates with the individual participants, providing up to date schedules, schedule status, and other participant selections of objective value ranges. The Multi-Objective Scheduler provides the evolutionary algorithm optimizer that evolves a population of candidate schedules towards the Pareto-optimal surface. The Application Map provides a transformation between decision variable values and domain-specific scheduling decisions as represented and evaluated in the Domain Scheduling Engine components. The Multi-Objective Scheduler supports parallel evaluations of schedules, which can frequently help speed the generation of a Pareto surface for participants. The Domain Scheduling Engine is the application-specific scheduling software that MUSE uses to evaluate candidate schedules. This evaluation utilizes the decision variable values, and can potentially perform internal conflict resolution or optimization steps on its own before returning a set of objective function values to the Multi-Objective Scheduler.

Visualization Tools

The example visualizations shown in this demonstration will be based on an application of MUSE to scheduling the James Webb Space Telescope (Giuliano and Johnston 2008). In this domain there are three objectives. To minimize gaps in the schedule, to minimize momentum build-up in telescope reaction wheels used to move the telescope, and to minimize observations which would be dropped as they missed their last opportunity to schedule.

The main focus of this demonstration is the Participant Trade Off GUI. The goal of this tool is to provide users of the system the ability to explore a Pareto-surface of potential solutions and to converge on acceptable solutions

for execution. A challenge for the MUSE system is visualizing Pareto-surfaces. The traditional approach is to display the surface as a series of X-Y trade-off plots. For example, Figure 2 illustrates a trade off surface for a JWST schedule run. Although, X-Y plots are intuitive to understand they have several problems. First, the number of plots grows geometrically as the number of objectives increases. Second, it is hard to connect a point in one plot with the corresponding points in other plots. An alternate view of the data is provided in Figure 3 by a parallel coordinate plot that solves the problems with X-Y plots. In this plot each solution is represented by a single colour coded line. The values are plotted horizontally on a normalized scale. Coordinate plots solve the problems with X-Y plots but can be unreadable with a large number of points on a Pareto-surface. The important point here is that no single view of the data is always best and that the interface needs to provide multiple views. With this goal the MUSE interface provides several additional features that allow the user to dynamically explore a Pareto-surface. First, MUSE provides a tabular view of the data that supports dynamic sorting (figure 5 top). Second, we provide a plot for each criteria that graphs the criteria values in order (Figure 5 bottom). Third, the user can select a solution in one plot and have the point highlighted in all of the plots (Figure 5). Each of the plots is linked so selection a solution or region in one plot highlights the corresponding solution or region in other plots. Ongoing MUSE development is exploring additional dynamic graphical capabilities such as the ability to collapse a N dimensional objective space to an N-1,2,… dimensional objective space.

Multi-Objective applications often have different constituents that are more or less concerned with specific mission objectives and criteria. For example, engineering staff may be more concerned with telescope lifetime issues such as momentum usage in JWST. In contrast science teams would be more concerned with not dropping observations (i.e. missing the last opportunity to schedule an observation). A goal of the MUSE system is to allow multiple constituents to converge on an acceptable region of the Pareto-surface. To this end the MUSE system models different users for an application (e.g. JWST engineering, JWST science operations). Users can login and select an active schedule interval to work on from a list of active intervals. For a schedule interval a user can select preferred regions of the solution space. Figure 5 shows the interface after the JWST engineering staff has entered a preference for a momentum build up. User preferences are stored on disk and can be viewed by other users. Figure 6 shows the interface run from the perspective of the JWST science operations staff after entering a preference for dropped observations. The interface shows solutions acceptable by both, one, and no users. The interface supports an administrative user who can make the final selection or override user preference if no agreement is available.

Conclusions and Future Work

Visualization tools for the MUSE system were described that allow multiple users to explore Pareto-surfaces. The tools provide multiple views of the data that can be configured to allow users to dynamically explore the search space. The tools allow multiple users with different priorities to specify preferences and to converge on a set of solutions acceptable to all parties. Future work on the system will add additional dynamic visualization features and will evaluate its use on domains with a higher number of objectives.

Acknowledgements

The research described in this paper was carried out at the Space Telescope Science Institute under the NASA Applied Information Systems Research Program grant number NNX07AV67G, and at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

Giuliano, M.E., and Johnston, M.D. 2008. Multi-Objective Evolutionary Algorithms for Scheduling the James Webb Space Telescope. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Sydney, Australia.

Johnston, M.D. and Giuliano, M.E, MUSE: The Multi-User Scheduling Environment for Multi-Objective Scheduling of Space Science Missions. In *IJCAI Workshop on Space Applications of AI*, Pasadena, CA.

JOHNSTON, M.D. AND GIULIANO, M. 2010. Multi-User Multi-Objective Scheduling for Space Science Missions. In *SpaceOps 2010*, Huntsville, AL.

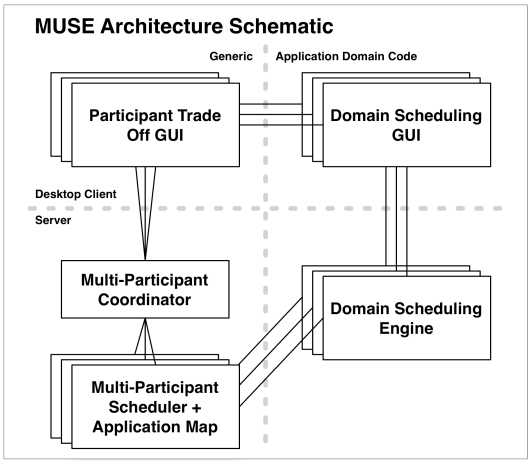

Figure 1: Multiple User Scheduling Environment (MUSE) system architecture

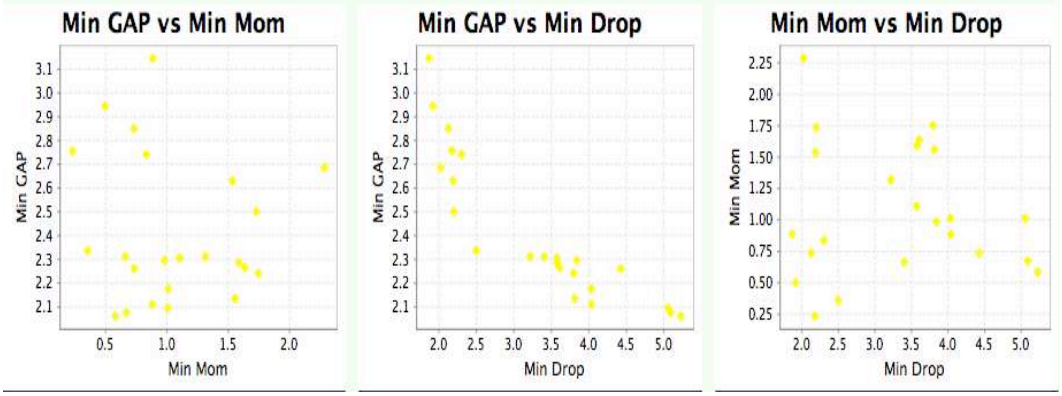

Figure 2: MUSE X-Y plots showing criteria trade-offs

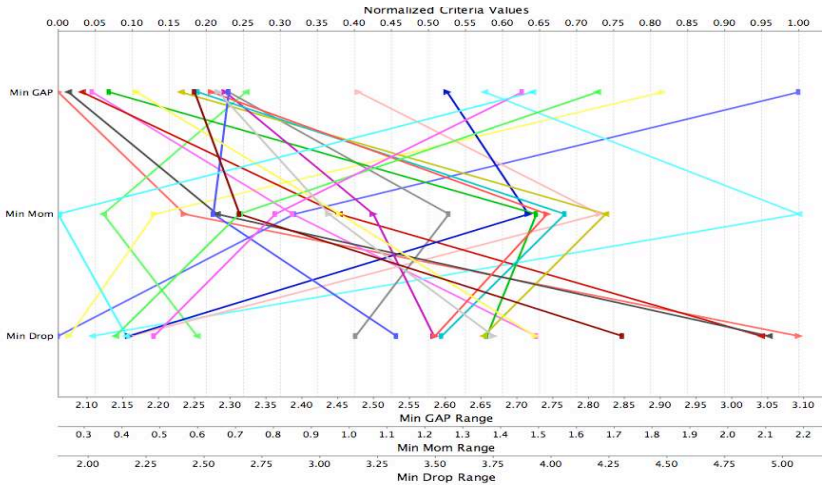

Figure 3: MUSE parallel coordinate plot

Index	Min GAP	Min Mom	Min Drop
0	2.06	0.5820426183...	5.23
1	3.145	0.8862978897...	1.87
2	2.335	0.3567022812...	2.5
3	2.945	0.4990168659...	1.92
4	2.11	0.8820475086...	4.04
5	2.685	2.2850866883...	2.025
6	2.5	1.7325837896...	2.2
7	2.31	1.3148033978...	3.22
8	2.095	1.0102924033...	5.0600000000...
9	2.63	1.5354204210...	2.19
10	2.1350000000...	1.5577325599...	3.815
11	2.24	1.7493877104...	3.8000000000...
12	2.305	1.1054975821...	3.575
13	2.265	1.6368651392...	3.61
14	2.075	0.6720651372...	5.095
15	2.285	1.5873128994...	3.58
16	2.31	0.6619029822...	3.4050000000...
17	2.85	0.7335429807...	2.13
18	2.1750000000...	1.0126939769...	4.035
19	2.74	0.8336212967...	2.305
20	2.755	0.2315510987...	2.18
21	2.295	0.9827801601...	3.845
22	2.2600000000...	0.7345708652...	4.43

Figure 4: MUSE tabular and index plots

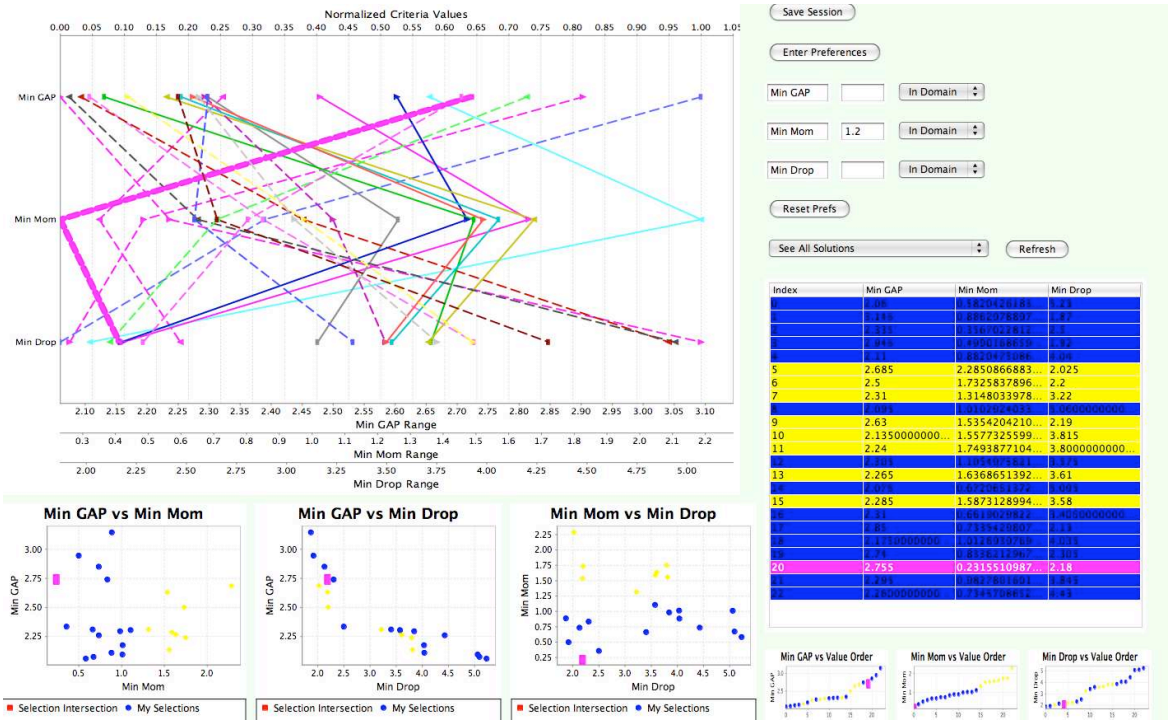

Figure 5: MUSE interface showing user selections

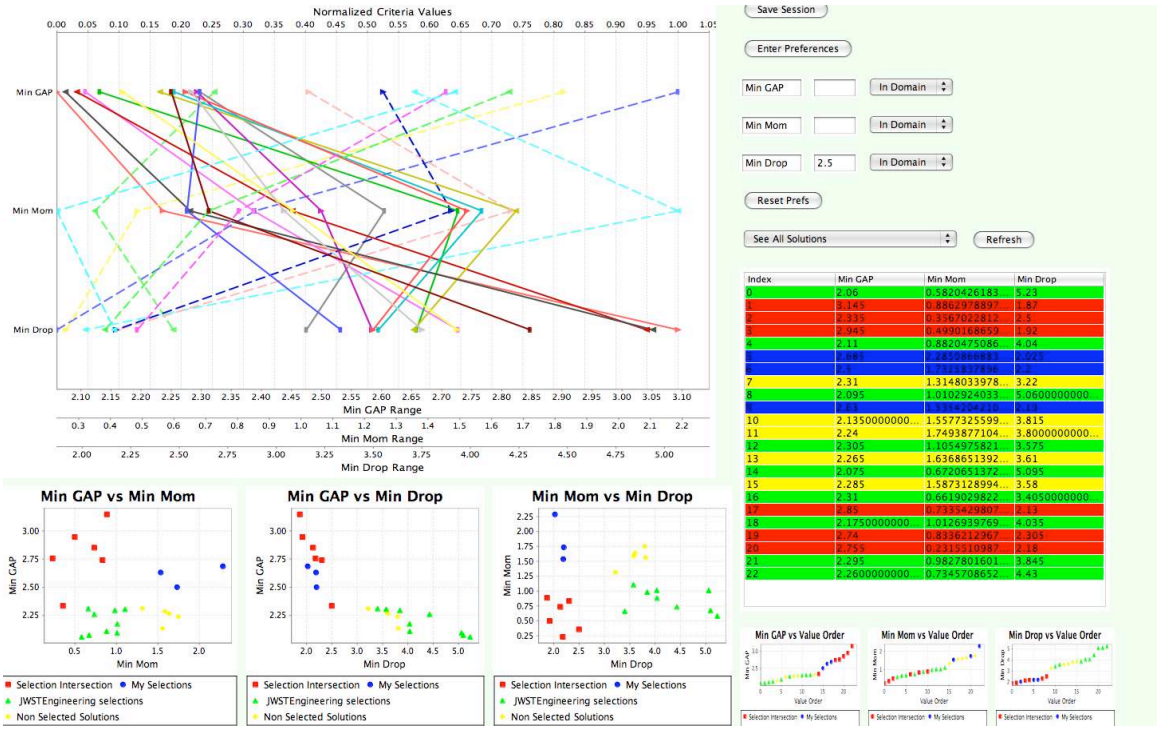

Figure 6: MUSE interface showing interactions between multiple user selections

The Scanalyzer Domain: Greenhouse Logistics as a Planning Problem

Malte Helmert
Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Georges-Köhler-Allee 52
79110 Freiburg, Germany
helmert@informatik.uni-freiburg.de

Hauke Lasinger
LemnaTec GmbH
Schumanstr, 18
52146 Würselen, Germany
hauke.lasinger@lemnatec.com

Abstract

We introduce the Scanalyzer planning domain, a domain for classical planning which models the problem of automatic greenhouse logistic management.

At its mathematical core, the Scanalyzer domain is a permutation problem with striking similarities to common search benchmarks such as Rubik's Cube or TopSpin. At the same time, it is also a real application domain, and efficient algorithms for the problem are of considerable practical interest.

The Scanalyzer domain was used as a benchmark for sequential planners at the last International Planning Competition. The competition results show that domain-independent automated planners can find solutions of comparable quality to those generated by specialized algorithms developed by domain experts, while being considerably more exible.

Real-Time Multi-Agent Planning and Scheduling in Dynamic Uncertain Domains *

Rajiv T. Maheswaran, Craig M. Rogers, Romeo Sanchez and Pedro Szekely
University of Southern California - Information Sciences Institute
4676 Admiralty Way #1001, Marina Del Rey, CA, USA

Abstract

Creating decision support systems to help people coordinate in the real world is difficult because it requires simultaneously addressing planning, scheduling, uncertainty and distribution. Generic AI approaches produce inadequate solutions because they cannot leverage the structure of domains and the intuition that end-users have for solving particular problem instances. We present a general approach where end-users can encode their intuition as guidance enabling the system to decompose large distributed problems into simpler problems that can be solved by traditional centralized AI techniques. Evaluations in field exercises with real users show that teams assisted by our multi-agent decision-support system outperform teams coordinating using radios.

Introduction

Teams of people need to coordinate in real-time in many dynamic and uncertain domains. Examples include disaster rescue, hospital triage, and military operations. It is possible to develop a plan *a priori* for these domains, but many parts must be left unspecified because people won't know exactly what needs to be done until they are executing the plan in the field. Additionally, requirements and tasks can evolve during execution.

Our work addresses a fundamental multi-agent systems endeavor of creating decision support systems that help humans perform better in real-time dynamic and uncertain domains. The technical challenges to compute good solutions for such domains have been well documented (Murphy 2004; Groen et al. 2007; Boutilier 1999). There are two main contributions in this paper: (1) we present a *generic methodology* for human guidance for planning and scheduling activities, and (2) we discuss an *extensive investigation* as to its usefulness in a thorough field exercise conducted by a third party.

In practice, it is possible to address specific domains with custom algorithms that use powerful heuristics to leverage the structures unique to that domain. These solutions are expensive to create as even these domains involve planning, uncertainty and distribution. The goal remains to develop *generic* approaches that produce good solutions that help human teams in many domains.

We introduce a new approach, STaC, based on defining collections of **S**ubteams each with **Ta**sks to perform and **C**onstraints on how they should be performed. The premise that people have good intuitions about how to solve problems in each domain and this approach both matches this intuition and can be matched to generic models of task allocation problems. The idea is to enable users to encode this intuition as guidance for the system and to use this guidance to vastly simplify the problems that the system needs to address.

The key to STaC is using the model and guidance to produce sufficiently smaller task structures that can be centralized so that a single agent can determine who does what, when and where with respect to these significantly simpler task structures. This mitigates the distribution challenge and enables using auxiliary solvers based on established AI techniques which produce good solutions at a smaller scale. These smaller task structures are solved independently assuming that the human guidance has addressed any significant dependencies. While this may not be the case in all domains, in many scenarios including ours, humans are far better at identifying effective structural decompositions than automated techniques.

STaC addresses tracking the dynamism in these task structures, the transitioning of agents assignment between these smaller task structures and the invocation of auxiliary solvers. Given that the task structures are treated independently and sufficiently small to be centralized, we call them sandbox reasoners. The sandbox reasoners required in each domain are different, so custom code must be written for each domain. However, the benefit of the approach is that sandbox reasoners are significantly simpler than the custom solvers required to produce a custom solution for a domain.

The paper is organized as follows. The next sections introduces the real-world domain where our approach was tested

*The work presented here is funded by the DARPA COORDINATORS Program under contract FA8750-05-C-0032. The U.S.Government is authorized to reproduce and distribute reports for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them. Approved for Public Release, Distribution Unlimited.

Figure 1: Field Exercise Images from Rome, New York, USA

followed by related work. We then describe the details of the STaC approach and the particular sandbox reasoners used in our example domain. We close with evaluation results, conclusions and directions for future work.

Field Exercises

The field exercises were based on a simulated disaster rescue domain. The challenge was to show that a human-team supported by intelligent agents could outperform a human team operating by themselves. The first two exercises were held in the city of Rome, New York, USA, and the second three were in Stanton Wood Park in Herndon, Virginia, USA. Images of the field exercise in Rome are shown in Figure 1 and a map of the sites and road network of Stanton Wood Park are shown in Figure 2. They were organized and evaluated by independent parties contracted by the DARPA (Defense Advanced Research Projects Agency) Coordinators program. The rules of the field exercise were created collaboratively by the teams building coordinator agents, the independent evaluation team, and subject matter experts. The specific instances or *scenarios* that comprised the test problems were chosen by the independent evaluation team.

Various locations were selected as *sites* and a feasible road network was constructed. If the site was *populated*, it could have injured people in either *critical* and *serious* condition. Populated sites would also have gas, power and water substations which may have been damaged. In addition, any site could have *facilities* such as a *hospital*, *clinic*, *warehouse*, *gas main station*, *power main station* and *water main station*. A team would obtain points by rescuing injured to hospitals or operational clinics (before a deadline associated with each injured person) and by repairing main stations and substations. The goal of a scenario was to accumulate as many points as possible before the scenario deadline.

The teams were composed of 8 field agents and 2 command agents. Each agent had a different set of skills. Three *specialists* in *gas*, *power* and *water* could perform *major* and *minor* repairs in their respective skill area. The *medical specialist* could load any type of injured person by themselves. The remaining four *survey specialists* could have any collection of skills involving minor repairs. The field agents could move throughout the field exercise area and perform actions. The command agents were located at a base where they helped to coordinate the activities of the team. The *Radio Team* communicated only with radios. Our *CSC Team* had ruggedized tablet computers on which our agents were loaded, in addition to radios. The tablets had cell modems and GPS.

Many outcomes were revealed during the game for which little or no likelihood information was given *a priori*, i.e., no probability distribution functions over outcomes. Teams did know the space of possible outcomes beforehand. A *survey for damage* at a main station or substation revealed the number and type of problems chosen from a set of known possible problems. A *survey for injured* at a populated site revealed the number, types and deadlines for the injured at that site. As the result of a survey, any team member might be injured, forcing them to go to an operational medical facility to recover before proceeding with any other action. A survey could also reveal that the vehicle of the agent doing the survey had failed and would require a vehicle repair before the agent could travel to any other site. While traveling, agents could encounter*road blocks* which could not be passed until fixed. Travel and repair times could vary and repairs could fail. These dynamic and uncertain events were planned parts of the exercise. In addition, the teams had to address uncertainties inherent in the environment, such as noisy radios, weather, and other activities in the public settings. Furthermore, most of these outcomes were only observable by the agent encountering the outcome.

The independent evaluation team chose the scenario from the space of possible exercises and informed the teams of the details below one day prior to the test: (1) the locations of populated sites and facilities, (2) the road network and ranges on travel times between sites, (3) a range for the total number of injured at each site, (4) the points for rescuing each type of injured, which could vary by type and site, (5) the points for repairing each substation or main station, which could vary by type and site, (6) potential problems after surveys for damage and corresponding repair options, (7) ranges on repair times, (8) likelihoods of failure for every repair activity, and (9) the skills of the survey specialist agents. The deadlines (for the scenario and injured) did not allow teams to do all possible repairs and rescues. The teams had one day to form a high-level strategy. The only element of uncertainty which could be modeled accurately with a probability density function was (8). When a team member completed a repair activity, they would call the evaluation team, which would report whether the repair was successful or a failure. The range in (3) was respected by the scenario designers, i.e., the number of injured did not fall outside the given range.

There were many rules and couplings that forced agents to coordinate. To do surveys, gas and power substations at the site had to be off, which required agents with those skills. Two agents had to be at the same location simultaneously to load a critically injured person or repair a road block. Repair options could involve multiple tasks and require two agents with certain skills to act in synchrony or in a particular sequence. Some repair options required kits

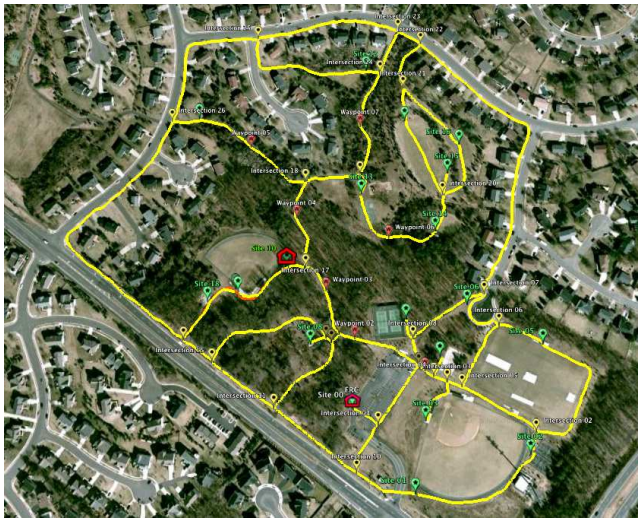

Figure 2: Stanton Woods Park, Herndon, Virginia, USA

which guaranteed their success, but kits were available only at warehouses. Agents could transport at most one entity, i.e, either a repair kit or a single casualty. A substation was considered repaired only if the corresponding main station was also repaired. A clinic was not operational until all substations at the site and all corresponding main stations were repaired. These are examples of rules that, along with the dynamism and uncertainty in outcomes mentioned earlier, created challenging real-time real-world distributed coordination problems.

The goal was to see if humans operating with radios and a multi-agent decision-support system could outperform humans operating with only radios. While some aspects of a real-world disaster scenario were abstracted, we believe the field exercises closely approximated the challenges of helping a human team solve difficult real-world problems.

Related Work

The STaC framework was developed during the DARPA Coordinators program. In the first two years, DARPA ran competitive evaluations on simulated scenarios, and CSC (Criticality-Sensitive Coordination), the underlying system behind the STaC framework, won such evaluations by considerable margins against two competing approaches based on Markov-Decision-Processes (MDPs) (Musliner et al. 2006) and Simple Temporal Networks (STNs) (Smith et al. 2007).

The MDP-based (Musliner et al. 2006) approach addressed the infeasibility of reasoning over the joint state space by setting the circumstance set to a subset of local state space that is reachable from the current local state, unrolling the state space by doing a greedy estimation of boundary values. It biased its local reward function on the commitments made by the agents during execution. However, such approximations lose critical information, exploring state spaces that are far from good distributed solutions.

The STN framework (Smith et al. 2007) addressed temporal uncertainty by using a time interval (instead of a point) as the circumstance that denoted feasible start times for a method to be executed. The system used *constraint propagation* to update the start intervals of the agents' activities during execution. A policy modification phase was triggered if execution was forced outside the given set of intervals. One of the problems of this approach is that agents tried to maintain consistency and optimize their local schedules, losing information that was needed to timely trigger policy modifications for their schedules.

We encoded scenarios of the field exercise as planning problems using PDDL (Planning Domain Definition Language) (Fox and Long 2006). The motivation was to identify to the extent to which current automated planning technology can address complex distributed, resource-driven, and uncertain domains. Unfortunately, this proved to be extremely difficult for state-of-the-art planning systems. From the set of planning systems tried, only LPG-TD (Gerevini et al. 2005), and SGPLAN (Chen, Wah, and Hsu 2006) solved a few simplified problems, after uncertainty, dynamism, non-determinism, resource-metrics, partial observability and deadlines were removed. Planners were unable to scale to more than 5 sites. LPG-TD produced solutions more efficiently but less optimally.

In general, mixed-initiative approaches where humans and software collaborate can often produce better solutions for complex problems. Mixed-initiative planning systems have been developed where users and software interact to construct plans. Users manipulate plan activities by removing or adding them during execution while minimizing the changes from a reference schedule (Ai-Chang et al. 2004; Hayes, Larson, and Ravinder 2005; Myers et al. 2003). Most of these systems are centralized, so humans and systems are fully aware of the entire plan, and of the consequences of updating it. In our scenario, agents (including humans) have subjective views of the world, and any decision may trigger many unknown global effects.

Multi-agent systems for disaster domains have been studied in the context of adjustable autonomy. The idea is to improve limited human situational awareness that reduces human effectiveness in directing agent teams by providing the flexibility to allow for multiple strategies to be applied. A software prototype, DEFACTO, was presented and tested on a simulated environment under some simplifications (e.g., no bandwidth limitations, reliable communications, omnipresence) (Schurr et al. 2005).

Conclusions and Future Work

Our 18-month experience working on a system to compete against radio teams in the field exercises provided evidence for the benefits of our approach. Our starting point was our generic CSC system developed during the previous two years to solve generic, synthetically generated problem instances specified in CTAEMS. Even though the synthetically generated problem instances were generated according to templates that combined "typical" coordination situations, the resulting problems were not understandable by humans. In contrast, the field exercise problems are more natural, and appeal to our lifetime of experience coordinating every day activities. Intuitions about space, distance, time, importance

and risk all came into play, enabling teams of humans to devise a sophisticated strategy with a few hours of brainstorming. It became obvious early on that the generic CSC system would not be able to produce solutions comparable to the desired sophisticated, coordinated behavior of human-produced strategies.

Our existing system had performed extremely well in Phase 2 by using our Predictability and Criticality Metrics (PCM) approach. In the PCM approach, the policy modifications that agents consider are limited to those that can be evaluated accurately through criticality metrics that capture global information. These policy modifications were simple and thus the reasoners that implemented them were simple too.

For the field exercises, we extended our approach so that policy modifications would be constrained using the guidance provided by the users. This guidance was in the form of a sequence of sites to visit. The system was left to make decisions that we believed it could evaluate accurately (e.g., how to perform repairs or rescue injured at a single site). The system relied on the TCR-set criticality metric to determine how to move agents along the list of guidance elements. The approach worked well. Our users outperformed the radio team because they were able to communicate their strategy to their agents, and the system optimized the execution of the strategy, adapting it to the dynamics of the environment.

The field exercises in Rome used a simpler language for specifying guidance. It had a single guidance group consisting of the entire set of agents. Also, it did not support constraints to control the capabilities within a guidance element. In that evaluation, our system remained competitive with the radio team, but lost in two out of the three scenarios.

The final language for guidance was inspired by our observations of the radio-team strategies, extensive discussions with subject matter experts and extensive numbers of simulations. We noted that while the human team could not execute a strategy as well as we could, the space of strategies that they were able to engage were far more sophisticated than ours. This led to the creation of a the more sophisticated formalism for capturing human strategic guidance.

We have taken the first step towards generic coordination technology that end-users can tailor to specific problem instances. The approach was validated in one domain thanks to the extensive and expensive evaluations carried out by the DARPA Coordinators program. In the future, we hope to be able to apply this approach to other application domains. One key area that needs to be investigated is extensions to allow human users to make guidance adjustments *during* execution. There are situations where a series of outcomes either invalidates an assumption when creating the *a priori* guidance or creates an opportunity to improve on that guidance. Addressing this requires the ability for human users to quickly and easily understand and modify the guidance while it is being executed. Even more advanced steps would be evaluating and ultimately generating appropriate online guidance modifications.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; jung Hsu, J. C.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. F. 2004. Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 19(1):8–12.

Boutilier, C. 1999. Multiagent systems: Challenges and opportunities for decision-theoretic planning. *AI Magazine* 20(4):35–43.

Chen, Y.; Wah, B.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in sgplan. *Journal of Artificial Intelligence Research (JAIR)* 26(1):323–369.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research (JAIR)* 27.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2005. Fast planning in domains with derived predicates: an approach based on rule-action graphs and local search. In *Proceedings of the 20th national conference on Artificial intelligence (AAAI)*, 1157–1162. AAAI Press.

Groen, F. C.; Spaan, M. T.; Kok, J. R.; and Pavlin, G. 2007. *Real World Multi-agent Systems: Information Sharing, Coordination and Planning*, volume 4363/2007 of *Lecture Notes in Computer Science. Logic, Language, and Computation*. Springer-Verlag.

Hayes, C.; Larson, A.; and Ravinder, U. 2005. Weasel: A mipas system to assist in military planning. In *ICAPS05 MIPAS Workshop (WS3)*.

Murphy, R. R. 2004. Human-robot interaction in rescue robotics. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 34(2):138–153.

Musliner, D. J.; Durfee, E. H.; Wu, J.; Dolgov, D. A.; Goldman, R. P.; and Boddy, M. S. 2006. Coordinated plan management using multiagent MDPs. In *Proceedings of the 2006 AAAI Spring Symposium on Distributed Plan and Schedule Management*.

Myers, K. L.; Jarvis, P. A.; Mabry, W.; Michael, T.; and Wolverton, J. 2003. A mixed-initiative framework for robust plan sketching. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*.

Schurr, N.; Marecki, J.; Scerri, P.; Lewis, J.; and Tambe, M. 2005. *The DEFACTO System: Coordinating human-agent teams for the future of disaster response*. Springer Press. chapter Programming multi-agent systems: Third international workshop.

Smith, S.; Gallagher, A. T.; Zimmerman, T. L.; Barbulescu, L.; and Rubinstein, Z. 2007. Distributed management of flexible times schedules. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2007)*.

Shopper: A System for Executing and Simulating Expressive Plans

John Maraist and **Robert P. Goldman**
SIFT, LLC
211 North First Street, Suite 300
Minneapolis, MN 55401-1480
{jmaraist,rpgoldman}@sift.info

Abstract

In this demo we present Shopper, a plan execution engine that facilitates experimental evaluation of plans. Shopper interprets the LTML plan language, which extends PDDL in two major ways: with more expressive control structures, and with support for semantic web services marked up in OWL-S. LTML's command structures include not only conventional ones such as branching, iteration, and procedure calls, but also features needed to handle HTN plans, such as precondition-filtered method choice. Unlike conventional programming languages, LTML supports interaction with the agent's belief store, so that its execution semantics line up with those assumed by planners. LTML actions extend PDDL actions in having outputs as well as effects, which means that they can support actions that sense the world; an important special case of this is semantic web services, which reveal information about a state hidden from the agent. To support experimentation as well as action in the real world, Shopper accommodates multiple, swappable implementations of its primitive action API. For example, one may interact with real web services through SOAP and WSDL, or with simulated web services through local procedure calls. In our demonstration we will show what LTML plans look like, relating them to their PDDL ancestors. We will show Shopper interpreting a plan whose individual steps are web service invocations. Then we will show how one may program simulated web services (or other actions) for use in experimentation, and show how Shopper operates with this simulation.

Planning for Data Mining Tool (PDM) Extended Abstract

Javier Ortiz and **Rubén Suárez** and **Tomás de la Rosa**
Susana Fernández and **Fernando Fernández** and **Daniel Borrajo**
Departamento de Informática
Universidad Carlos III de Madrid,
Avda. de la Universidad 30
28911 Leganés (Madrid), Spain

David Manzano
Ericsson Research Spain
Madrid, Spain

Introduction

We present a tool, PDM (Planning for Data Mining), based on Automated Planning that helps users (non necessarily experts on data mining) to perform DM (Data Mining) tasks. The starting point is a definition of the DM task to be carried out and the output is a set of plans that are executed in a DM tool to obtain a set of models and statistics. Plans are data-mining knowledge flows, i.e. sequences of DM actions that should be executed over the initial datasets to obtain the final models. However, the number of feasible plans that solve the same DM task is huge making necessary to rank them by some criterion. In a first approach, the ranking is performed following some expert estimations on the desired mining-results of the DM actions. Afterwards, these estimations are improved using machine learning techniques. In order to define the DM task, we use emerging standards, such as PMML (Predictive Model Markup Language). PMML is the leading standard for statistical and DM models and supported by over 20 vendors and organizations. With PMML, it is straightforward to develop a model on one system using one application and deploy the model on another system using another application. The PMML file is automatically translated into a planning problem described in PDDL2.1. So, any state-of-the art planner can be used to generate a plan (or plans), i.e. the sequence of DM actions that should be executed over the initial dataset to obtain the final model. Each plan or knowledge flow is executed by a machine learning engine. In our case, we employ one of the most used DM tools, WEKA (Witten and Frank 2005). In WEKA, knowledge flows are described as files with a specific format, KFML, and datasets are described as ARFF (Attribute-Relation File Format) files. The results of the DM process can be evaluated, and new plans may be requested to the planning system.

Background

This section describes two of the three languages used in this work and the files used in the learning component. First, we describe PMML (Predictive Model Markup Language), an XML based language for DM. Then, we describe KFML (Knowledge Flow for Machine Learning), another XML based language to represent DM knowledge flows for the WEKA tool (Witten and Frank 2005). The third language used in PDM, PDDL (Planning Domain Definition Language), is well known in the planning community. Finally we describe the type of files used in the learning process.

The Predictive Model Markup Language (PMML)

PMML is an XML-based markup language developed by the Data Mining Group (DMG) to provide a way for applications to define models related to predictive analytics and DM and to share those models between PMML-compliant applications. [1] It is composed of five main parts:

- The header contains general information about the file, like the PMML version, date, etc.
- The data dictionary defines the meta-data, or the description of the input data or learning examples.
- The transformation dictionary defines the applicable functions over the input data, like flattening, aggregation, average or normalization among many others. This knowledge defines the actions that can be applied over the data in the first step of the mining process.
- The models contain the definition of the DM models.
- The mining build task describes the configuration of the training task that will produce the model instance. This mining build task can be seen as the description of the sequence of actions executed to obtain the model. From the perspective of planning, it can be seen as a plan. This plan would include the sequence of DM actions that should be executed over the initial dataset to obtain the final model.

WEKA and the Knowledge Flow Files (KFML)

WEKA (Witten and Frank 2005) is a collection of machine learning algorithms to perform DM tasks. It includes all the software components needed in a DM process, from data loading and filtering to advanced machine learning algorithms for classification, regression, etc. It also includes many interesting functionalities, like graphical visualization of the results. WEKA offers two different usages. The first one is using directly the WEKA API in Java. The second

[1] See www.dmg.org for further information.

one consists of using the graphical tools offered. The tool included in WEKA we use is the Knowledge Flow. WEKA Knowledge Flow is a data-flow inspired interface to WEKA components. It allows to build a knowledge flow for processing and analyzing data. Such knowledge flow can include most of WEKA functionalities: load data, prepare data for cross-validation evaluation, apply filters, apply learning algorithms, show the results graphically or in a text window, etc. Knowledge flows are stored in KFML files, that can be given also as input to WEKA.

A KFML file is an XML file including two sections. The first one defines all the components involved in the knowledge flow, as data file loaders, filters, learning algorithms, or evaluators. The second one enumerates the links among the components, i.e. it defines how the data flows in the DM process, or how to connect the output of a component with the input of other components. WEKA Knowledge Flow allows loading, graphically editing, executing and saving KFML files. A knowledge flow can be seen as the sequence of steps that must be performed to execute a DM process.

Attribute-Relation File Format (ARFF)

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed for use with the WEKA machine learning software.[2] ARFF files have two distinct sections. The first section is the Header information, which is followed by the Data information. The Header of the ARFF file contains the name of the relation (name of the DM task), a list of attributes (potentially including the class), and their types. The ARFF Data section of the file contains the actual instances, described in terms of the values of the attributes.

The Planning for Data Mining (PDM) Tool

Figure 1 shows the PDM architecture of the implemented system. There are four main modules; each one can be hosted in a different computer connected through a network: *Client*, *Control*, *Datamining* and *Planner*. We have used the Java RMI (Remote Method Invocation) technology that enables communication between different servers running JVM's (Java Virtual Machine). The planner incorporated in the architecture is SAYPHI (De la Rosa, García-Olaya, and Borrajo 2007) and the DM Tool is WEKA (Witten and Frank 2005). However, given that we are using standard languages other planners and/or DM tools could have been used.

The *Client* module offers an interface that provides access to all the application functionalities. It generates a PMML file from a high level description of the DM task specified by the user using the interface. Then, it sends the PMML description to the *Control* module.

The *Control* module interconnects all modules and performs the required translations. The translations needed are: from PMML to PDDL, `PMML2PDDL`; and from a PDDL plan to KFML, `Plan2KFML`. The input to the module is the DM task together with the dataset. First, the `PMML2PDDL`

[2]See www.cs.waikato.ac.nz/ ml/weka/arff.html for further information.

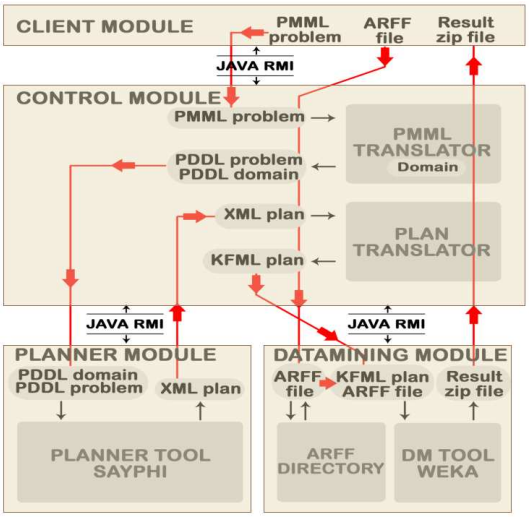

Figure 1: Overview of the PDM architecture.

translator generates the PDDL problem file from the PMML file. Then, the planner is executed to solve the translated problem. The returned set of plans is translated to several KFML files. Finally, the DM Tool is executed to process and run the translated KFML files. The *result* is a compressed file containing a set of directories, one for each plan. Each directory contains the model generated by the DM Tool, the statistics related to the evaluation of the model, the plans generated by the planner, and the corresponding DM workflow in KFML. This results are used to create an ARFF file. Such ARFF file let the system learn the estimations given by experts.

The *Datamining* module permits the execution of DM tasks in the WEKA DM Tool through Knowledge Flow plans. It can obtain the model output and the statistics generated as a result of the Knowledge Flow execution. This module also contains an *ARFF* directory for managing the storage of the datasets that are necessary for the WEKA executions. The input to the module is a KFML file and the output is a compressed file. The output is the model generated by the plan and the statistics related to the evaluation of the model.

The *Planning* module receives each problem and domain in PDDL format. It returns a set of plans in XML format ready for the conversion to a KFML format. Currently, planning tasks are solved by the SAYPHI planner (De la Rosa, García-Olaya, and Borrajo 2007), but the architecture could use any other planner that supports fluents, conditional effects and metrics. We have used SAYPHI because it: i) supports an extensive subset of PDDL; and ii) incorporates several search algorithms able to deal with quality metrics.

Building the models from DM Tasks in PDDL

The main challenge of our approach is how to model DM tasks by means of Automated Planning (AP). As we said, an AP task is defined by two files, the domain definition and

the problem description, whereas the DM task is defined by the PMML file.

The PMML file description

The header of the PMML file contains the following information:

- The DM goal. It can be classification, regression or clustering.
- The dataset location and size.
- The hard and soft constraints of the user. An example of hard constraint is obtaining an error lower than a given threshold, whereas minimizing the total execution time is an example of soft constraint or preference. We handle preferences and hard constraints over: i) *exec-time*, for minimizing/constraining the execution time, ii) *percentage-incorrect*, for minimizing/constraining the classification error; iii) *mean-absolute-error*, for minimizing/constraining the mean absolute error in regression tasks and clustering, and iv) *unreadability*, for maximizing the understandability of the learned model (we transform maximizing the understandability of the learned model for minimizing/constraining *unreadability*).

The data dictionary includes one field for each attribute in the dataset. The transformation dictionary includes one function for each possible filter the user wants to apply over the input data. Finally, the model part contains the definition of the DM models. The user can specify the same model but with different parameters. In general, for most DM tasks, a user would include in the PMML file the full set of WEKA DM techniques and some common settings for their parameters.

The PDDL Domain description

The PDDL domain file contains the description of all the possible DM tasks (transformations, training, test, visualization, ...). Each DM task is represented as a domain action. The PDDL problem files contain information for a specific dataset (i.e. dataset schema, the suitable transformation for the dataset, the planning goals, the user-defined metric, etc.). Domain predicates define the state space containing static information (i.e. possible transformations, available training or evaluation tasks, etc.) and dynamic information that changes during the execution of all DM tasks (e.g. adding the fact that the dataset has already been pre-processed or evaluated). The functions allow us to define thresholds for different kinds of DM features (e.g. error, execution time threshold, or understandability of a model) and to store the values updated during the execution (e.g. total estimated error, execution time, or understandability).

Compilation of a PMML into a PDDL Problem

The `PMML2PDDL` translator automatically converts parts of a PMML file with the DM task information into a PDDL problem file. The translator uses expert knowledge to define some important planning information like the execution time of the DM tasks, the classification error for classification models or the mean absolute error in regression tasks and clustering and the understandability of the each model. The problem file together with a domain file, that is fixed for all the DM tasks, are the inputs to the planner. The problem file contains the particular data for each DM episode, including the dataset description, the transformations and models available for that problem, and the possible preferences and constraints of the user.

Planning for DM Tasks

SAYPHI solves the planning task depending on the metric specified in the PMML file. SAYPHI includes a collection of search algorithms and domain-independent heuristics. Here, we use Best-first Search with the relaxed planning graph heuristic of FF (Hoffmann and Nebel 2001). Given that the heuristic is not admissible, it does not guarantee to find the best solution. Also, it is an open problem to assign the exact cost estimations (in terms of accuracy, time to learn, or understandability) to planning (DM) actions. So, once it finds a solution, it continues exploring nodes in order to find multiple solutions. Probably, the best models according to the planner are not necessarily the best models according to the user due to the estimation of the action cost explained below. Therefore, diversity is the only way to avoid this problem.

The Planner module outputs all the generated plans encoded in an XML file. `Plan2KFML` translates each plan into a KFML file, so it can be executed by the WEKA Knowledge Flow. The translator generates as output a new KFML file with an equivalent plan plus some extra actions. Each action in the PDDL domain corresponds to one or many WEKA components. Therefore, the translator writes for each action in the plan the corresponding set of XML tags that represent the WEKA component. Finally, the translator adds some extra components in order to save the information generated during the execution. That information is composed of the learned models and statistical information as the execution time, accuracy, ...

Learning

As defined above, the PDM architecture uses expert knowledge to define some important planning information, like the time required to build an specific model, or the estimated accuracy of the resulting model. Initially, these values are defined by an expert. However, those estimations can be far from correct values, since they are hard to define. Also, it can become difficult to provide those values under all possible uses of the techniques and the different domains.

The goal of the learning component is to automatically acquire all those estimations from the experience of previous DM processes. The data flow of this component is described in Figure 2.

The main steps of this flow are:

1. Gathering DM Results: the goal is to gather DM experience from previous DM processes. All the information is stored in an *ARFF* file. For a given DM process, the following information is stored:
   - Information about the data set: number of instances of the data set, number of attributes of the data set, number of continuous attributes, etc.

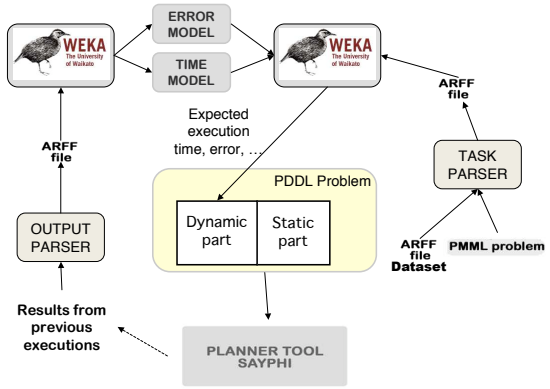

Figure 2: Learning flow in the PDM architecture.

- Information about the model to build: the type of model (association, clustering, general regression, neural network, tree, etc.), the algorithm used to learn the model (RBFNetwork, J48, etc.), the type of function (classification, regression, clustering), the learning parameters, etc.
- Information about the results obtained: type of evaluation (split, cross validation, training set), time to build the model, accuracy, mean squared error, etc.
- The plan that represents the DM workflow, and that has been executed to obtain the model

2. Model generation: the information obtained in the previous step is used to learn prediction models. The functions to learn are time, accuracy and SME (in Figure 2, error and time models. These models can be generated with the WEKA tool, as shown in the figure.

3. Given a new data set, a model, a function, and a learning algorithm, and using the models generated in the previous step, we obtain a prediction of the learning time, accuracy and SME that will be obtained if we perform a new DM process. These estimations are included in the PDDL file, so they are used when planning new DM processes. Figure 3 shows an example of how the fluents of the dynamic part of the PDDL problem file are updated. In the figure, the exec-time of a tree model and the support vector machine model are updated, among others.

There are two ways to update the PDDL problem file with these estimations: off-line and on-line. Off-line updates require to obtain information of many DM processes, use the execution information to build the models, and employ these models to update the PDDL problem file, which will stay fixed in the future. On-line updates assume that, while new data-mining processes are executed, new learning examples are obtained, so the models can be dynamically updated, and the PMML problem file is continuously updated.

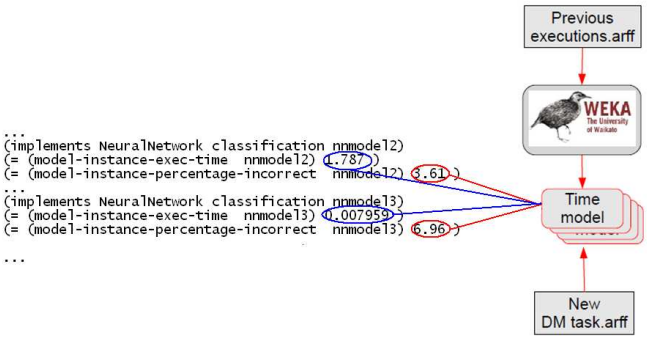

Figure 3: Learning example in the PDM architecture.

References

De la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2007. Using cases utility for heuristic planning improvement. In *Case-Based Reasoning Research and Development: Proceedings of the 7th International Conference on Case-Based Reasoning*, 137–148. Belfast, Northern Ireland, UK: Springer Verlag.

Hoffmann, J., and Nebel, B. 2001. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd Edition, Morgan Kaufmann.

Conformant Planners: Approximation vs. Representation

Son Thanh To, Dang-Vien Tran, Hoang-Khoi Nguyen, Tran Cao Son, Enrico Pontelli
Knowledge representation, Logic, and Advanced Programming Laboratory
Computer Science Department
New Mexico State University
Las Cruces, NM 88003

Abstract

This demonstration presents three conformant planners: CPA, DNF, and CNF, whose performances are comparable to those of state-of-the-art conformant planners. All three are best-first search and progression-based planners. CPA, a representative of approximation-based planners, was the recipient of the *Best Non-Observable Non-Deterministic Planner Award* at IPC-2008. DNF and CNF are complete planners which employ different representations for belief states and perform better than CPA in several domains. DNF uses DNF-formulae which are minimal with respect to set inclusion, while CNF uses CNF-formulae which are minimal with respect to subsumption. The key difference between these two planners and CPA lies in that they implement an algorithm for guaranteeing completeness of the planner only when needed, while CPA does so before the search for a plan starts. The heuristics employed by the three aforementioned planners are combinations of two well-know heuristics used in conformant planning: the size of the belief state and the number of satisfied subgoals. The demonstration also presents various techniques that contribute to the performance and scalability of these planners. Lessons learned during the development of these planners are discussed.

Introduction

Conformant planners deal with planning problems with uncertainty about the initial states. The following issues are key to the development of a conformant planner:

- A formalization of actions in presence of incomplete information; and
- A belief state representation and a good heuristic function.

The first item is important for the correctness of the planner, as it provides the theoretical foundations for the planner to progress in the presence of incomplete information. The second item is critical to the performance and scalability of the planner, since the complexity of the problem of computing the successor belief state is, in general, computationally expensive.

In this demonstration, we introduce three conformant planners, CPA, DNF, and CNF. These planners employ different approaches to searching for a solutions. CPA searches for a solution in the space of sets of partial states instead of the space of belief states (Son et al. 2005). CPA uses DNF-formulae to represent a set of partial states and employs several techniques to reduce the size of the initial belief state. CPA was the recipient of the *Best Non-Observable Non-Deterministic Planner Award* at IPC-2008. The exceptional performance of CPA led us to study the impact of the belief state representation on the performance of complete conformant planners. This investigation resulted in the development of two planners, DNF and CNF, which use DNF- and CNF-formulae, respectively, in encoding belief states.

We begin with a short review of the background of conformant planning. We then discuss the basic concepts used in the development of the planners. Afterwards, we describe the organization of the systems.

Background: Conformant Planning

A *planning problem* is described by a tuple $P = \langle F, O, I, G\rangle$, where F is a set of propositions, O is a set of actions, I describes the initial state of the world, and G describes the goal. A *literal* is either a proposition $p \in F$ or its negation $\neg p$. $\bar{\ell}$ denotes the complement of a literal ℓ—i.e., $\bar{\ell} = \neg\ell$, where $\neg\neg p = p$ for $p \in F$. For a set of literals L, $\overline{L} = \{\bar{\ell} \mid \ell \in L\}$. A conjunction of literals is often viewed as the set of its literals.

A set of literals X is *consistent* if there exists no $p \in F$ such that $\{p, \neg p\} \subseteq X$. A set of literals X is *complete* if for each $p \in F$, $\{p, \neg p\} \cap X \neq \emptyset$. A *state* s is a consistent and *complete* set of literals. A *belief state* is a set of states.

Each action a in O is associated with a precondition ϕ (denoted by $pre(a)$) and a set of conditional effects of the form $\psi \rightarrow \ell$ (also denoted by $a : \psi \rightarrow \ell$), where ϕ and ψ are sets of literals and ℓ is a literal.

A state s satisfies a literal ℓ, denoted by $s \models \ell$, if $\ell \in s$. s satisfies a conjunction of literals X, denoted by $s \models X$, if it satisfies every literal belonging to X. The satisfaction of a formula in a state is defined in the usual way. Likewise, a belief state S satisfies a literal ℓ, denoted by $S \models \ell$, if $s \models \ell$ for every $s \in S$. S satisfies a conjunction of literals X, denoted by $S \models X$, if $s \models X$ for every $s \in S$.

Given a state s, an action a is *executable* in s if $s \models pre(a)$. The effect of executing a in s is

$$e(a, s) = \{\ell \mid \exists(a : \psi \rightarrow \ell).\ s \models \psi\}$$

The transition function, denoted by Φ, in the planning do-

main of P is defined by

$$\Phi(a,s) = \begin{cases} s \setminus \overline{e(a,s)} \cup e(a,s) & s \models pre(a) \\ \bot & \text{otherwise} \end{cases} \quad (1)$$

where $\bot$ denotes a fail state.

Φ is extended to define $\widehat{\Phi}$, a transition function which maps sequences of actions and belief states to belief states for reasoning about the effects of plans. Let S be a belief state. We say that an action a is executable in a belief state S if it is executable in every state belonging to S. Let $\alpha_n = [a_1, \ldots, a_n]$ be a sequence of actions and $\alpha_i = [a_1, \ldots, a_i]$:

- If $n = 0$ then $\widehat{\Phi}([\,], S) = S$;
- If $n > 0$ then
  - if $\widehat{\Phi}(\alpha_{n-1}, S) = \bot$ or a_n is not executable in $\widehat{\Phi}(\alpha_{n-1}, S)$, then $\widehat{\Phi}(\alpha_n, S) = \bot$;
  - if $\widehat{\Phi}(\alpha_{n-1}, S) \neq \bot$ and a_n is executable in $\widehat{\Phi}(\alpha_{n-1}, S)$ then $\widehat{\Phi}(\alpha_n, S) = \{\Phi(a_n, s') \mid s' \in \widehat{\Phi}(\alpha_{n-1}, S)\}$.

The initial state of the world I is a belief state and is represented by a formula. In our investigation, we consider I to be a conjunction of literals, `oneof` statements and `or` statements—where a `oneof` statement (`or` statement) represents an exclusive-or (resp. logical or) of its components. By S_I we denote the set of all states satisfying I. Typically, the goal description G can contain literals and `or` statements.

A sequence of actions $\alpha = [a_1, \ldots, a_n]$ is a solution of P if $\widehat{\Phi}(\alpha, S_I)$ satisfies G. In this paper, we will denote with C_a the set of conditional effects of an action a.

Approximation-Based Planning

The approach to approximation-based planning adopted in CPA relies on the 0-approximation semantics for reasoning about effects of actions in presence of incomplete information about the initial state (Son and Baral 2001). Intuitively, the approach (*i*) replaces a belief state by a *partial state*, which is a set of fluent literals; and (*ii*) specifies how to compute the successor partial state, i.e., the result of executing an action in a given partial state. This is appealing for conformant planning, since it lowers the complexity of conformant planning (Baral, Kreinovich, and Trejo 2000). It is characterized by a function (Φ_A) that maps an action and a partial state to a partial state. Given a partial state δ, the possible effects of a in δ are given by

$$pc_a(\delta) = \{l \mid (\psi \rightarrow l) \in C_a, \overline{\psi} \cap \delta = \emptyset\}. \quad (2)$$

The successor partial state from the execution of a in δ is defined by $\Phi_A(a,\delta) = (\delta \cup e(a,\delta)) \setminus \overline{pc_a(\delta)}$ if a is executable in δ; and $\Phi_A(a,\delta) = \bot$, otherwise. This function is then extended to define $\widehat{\Phi_A}$, similarly to $\hat{\Phi}$, to reason about plans.

Observe that Φ_A coincides with Φ under complete information, i.e., $\Phi_A(a,s) = \Phi(a,s)$ for every state s. However, Φ_A can be incomplete. For example, given a planning problem P_1 with the set of propositions $\{f, g, h\}$, the initial state $I = \emptyset$, the set of actions $O = \{a : f \wedge g \rightarrow h, a : f \wedge \neg g \rightarrow h, b : g \rightarrow f, b : \neg g \rightarrow f\}$, and the goal $G = \{h\}$. $\Phi_A(b, \{\emptyset\}) = \{\emptyset\}$, i.e., Φ_A will answer the query whether f will be true after the execution of b in the initial state with 'No' whereas Φ will say 'Yes.'

To guarantee completeness, CPA exploits the completeness condition in (Son and Tu 2006) to identify a minimal set of initial partial states and searches for solutions in the space of sets of partial states, called *cs-states*.

DNF and CNF

The DNF planner uses DNF-formulae to represent belief states and employs the transition function Φ_{DNF} in its progression. Φ_{DNF} relies on an algorithm for splitting a partial state δ into a *minimal* set of partial states Δ (with respect to set inclusions among members) such that every precondition of a given action a is either true or false in each member of Δ. The details of Φ_{DNF} can be found in (To, Pontelli, and Son 2009).

For example,given the planning problem P_1 described in the previous subsection, the behavior of Φ_{DNF} is as follows. The initial belief state will be represented by the DNF-formula $\Delta = \{\emptyset\}$. $\Phi_{DNF}(b, \Delta)$ will begin with the realization that Δ will need to be split into $\Delta_1 = \{\{g\}, \{\neg g\}\}$. The result of executing b in Δ will then be $\Delta_2 = \{\{f, g\}, \{f, \neg g\}\}$. Executing a in Δ_2 does not require any splitting and results in $\Delta_3 = \{\{f, g, h\}, \{f, \neg g, h\}\}$.

Observe that the initial belief state consists of eight states (all possible states of the problem) and the DNF-formula after the splitting for b contains only two elements. This representation allows for an efficient computation of the successor belief state—i.e., $\Phi_{DNF}(a, \Delta)$ can be computed in polynomial time in the size of Δ, under a reasonable assumption that the number of effects of each action is bounded, for every action a and DNF-formula Δ. It is worth to mention that the belief state representation of DNF is similar to that of CPA. In this sense, the key distinction between DNF and CPA lies in that DNF computes the set of partial states needed for guaranteeing the completeness of the planner only when needed, while CPA computes it before the search process starts. Both planners, however, still suffer from the possible huge size of the initial state.

To address the problem of the size of the initial state faced by DNF and CPA, the CNF planner uses CNF-formulae, represented as a set of clauses and *minimal* with respect to subsumption and unit propagation, to represent belief states. The transition function Φ_{CNF} of CNF is similar to Φ_{DNF}, in that it also relies on an algorithm for splitting a formula φ into a set of CNF-formulae, $enb(a, \varphi)$, such that the effects of the action a on φ can be determined. For example, Φ_{CNF} behaves almost as Φ_{DNF} with respect to the problem P_1, as the initial state $\emptyset$ is splitted into two clauses $\{g\}$ and $\{\neg g\}$, enabling the execution of b to achieve f from the initial state. In this sense, the key difference between DNF and CNF lies in their use of different belief state representations. While the computation of Φ_{CNF} is more complex than that of Φ_{DNF}, its representation of the initial state is more compact. This representation pays off when the size of the initial state is huge. This representation also allows for the application of a new technique, called relaxation of `oneof` statements, which allows CNF to solve all instances of the `coin`-domain. To the best of our knowledge, CNF

is the only planner can deal with the instances `coins-21` to `coins-30`, whose initial belief states contain more than 10^6 states. Precise definition of Φ_{CNF} can be found in (To, Son, and Pontelli 2010).

Analysis and Simplifications

The analysis and simplification techniques implemented in the three planners help simplify the planning instances by reducing the number of actions and propositions. It also contains, for each representation, a technique that reduces the size of the initial cs-state (or belief state). These techniques briefly discussed next.

Basic Simplifications: We consider two well-known basic steps: *forward reachability* and *goal relevance*. Several planners implement these two steps.

Forward reachability is used to detect: *(i)* propositions whose truth value cannot be affected by the actions in the problem specification (w.r.t. the initial state); *(ii)* actions whose execution cannot be triggered w.r.t. the given initial state. This process can be modeled as a fixpoint computation. Goal relevance proceeds in a similar manner, by detecting actions that are relevant to the achievement of the goal.

Combination of `oneof` *Statements:* `oneof` statements are used to specify the uncertainty about some propositions and/or mutual exclusion between propositions. The number of the `oneof` statements and their size (the size of an `oneof` statements is the number of its elements) determine the size of the initial cs-state.

The idea of the combination of `oneof` statements technique is based on the *non*-interaction between actions and propositions in different sub-problems of a conformant planning problem. This idea is best illustrated with a simple example.

Let us consider the planning problem P_2 with the set of propositions $\{f, g, h, p, i, j\}$, the initial state $I = \{\texttt{oneof}(f, g), \texttt{oneof}(h, p), \neg i, \neg j\}$, the set of actions $O = \{\ a : f \rightarrow i \quad c : h \rightarrow j \quad b : g \rightarrow i \quad d : p \rightarrow j\ \}$, and the goal $G = i \wedge j$. Here, a causes i to be true if f is true; c causes j to be true if h is true; b causes i to be true if g is true; and d causes j to be true if p is true.

It is easy to see that the sequence $\alpha = [a, b, c, d]$ is a solution of P_2. Furthermore, the search should start from the cs-state consisting of the four states:

$$\{f, \neg g, h, \neg p, \neg i, \neg j\} \quad \{\neg f, g, h, \neg p, \neg i, \neg j\}$$
$$\{f, \neg g, \neg h, p, \neg i, \neg j\} \quad \{\neg f, g, \neg h, p, \neg i, \neg j\}$$

Let P_2' be the problem obtained from P_2 by replacing I with I', where $I' = \{\texttt{oneof}(f \wedge h, g \wedge p), \neg i, \neg j\}$.

We can see that α is also a solution of P_2'. Furthermore, each solution of P_2' is a solution of P_2. This transformation is interesting since the initial cs-state now consists only of two states: $\{f, \neg g, h, \neg p, \neg i, \neg j\}$ and $\{\neg f, g, \neg h, p, \neg i, \neg j\}$. In other words, the number of states in the initial belief state (or initial cs-state) that a conformant planner has to consider in P_2' is 2, while it is 4 in P_2. This transformation is possible because the set of actions that are "activated" by f and g is disjoint from the set of actions that are "activated" by h and p, i.e., $preact(\{f, g\}) \cap preact(\{h, p\}) = \emptyset$ where $preact(\delta)$ we denote the set of actions depending on δ.

Using this technique, many `oneof` statements can be combined into one, yielding several order of magnitudes reduction in the size of the initial cs-state.

Goal Splitting: The key idea is that if a problem P contains a subgoal whose truth value cannot be negated by the actions used to reach the other goals, then the problem can be decomposed into smaller problems with different goals, whose solutions can be combined to create a solution of the original problem. This technique can be seen as a variation of the goal ordering technique in (Hoffmann, Porteous, and Sebastia 2004) and relies on the notion of dependence proposed in (Son and Tu 2006).

Relaxation of `oneof` *Statements:* In contrast to the combination of `oneof` statements technique, a relaxation of an `oneof` statement increases the number of the states in the initial belief state. More precisely, the relaxation of a `oneof` statement $\texttt{oneof}(l_1, \ldots, l_k)$ replaces it with an `or` statement $\texttt{or}(l_1, \ldots, l_k)$. Conditions for the soundness of the transformation have been identified. This technique also relies on the non-interaction between actions and propositions in these `oneof` statements. Let us illustrate this with a simple example.

Let consider the planning problem P_3 with the set of propositions $\{f, g, i, j\}$, the initial state $I = \{\texttt{oneof}(f, g), \neg i, \neg j\}$, the set of actions $O = \{\ a : f \wedge \neg i \rightarrow i, b : g \wedge \neg j \rightarrow j\ \}$, and the goal $G = i \wedge j$. Any solution of the problem P_3' with the same set of propositions, the initial state $I' = \{\texttt{or}(f, g), \neg i, \neg j\}$, O, and G will be a solution of P_3.

Observe that this technique increases the number of states in the initial belief state. However, the size of the CNF formula representing the relaxation will be smaller compared to the size of the CNF formula representing the original belief state.

Heuristics in CPA

The heuristics used in CPA are the combination of the following well-known heuristics.

- *The cardinality heuristic:* we prefer cs-states that have a smaller cardinality. In other words, $h_{card}(\Sigma) = |\Sigma|$ where Σ is a cs-state. Note that we use this heuristic in a forward fashion, and hence, is different from its use in (Bertoli, Cimatti, and Roveri 2001; Bryce and Kambhampati 2004). The intuition behinds this is that planning with complete information is "easier" than planning with incomplete information, and a lower cardinality implies a lower degree of uncertainty.
- *The number of satisfied subgoals:* denoted by $h_{goal}(\Sigma)$.

CPA uses the combination: $h_{cs}(\Sigma) = (h_{card}(\Sigma), h_{goal}(\Sigma))$ with lexicographic ordering. It gives preference to the cs-states with a lower degree of uncertainty, i.e., cs-states that have a smaller cardinality. If the cardinality of two cs-states does not differ, then the heuristics gives preference to those cs-states that maximize the number of satisfied subgoal.

Heuristics in DNF and CNF

Given a DNF-state Δ, the heuristic function used in DNF is a combination of the following three values (along with a lexicographic ordering):

- $h_{goal}(\Delta)$: the number of subgoals satisfied by Δ.
- $h_{card}(\Delta)$: the cardinality of Δ.
- $h_{dis}(\Delta)$: the *square distance* of Δ to the goal, defined by

$$h_{dis}(\Delta) = \Sigma_{\delta \in \Delta}(|G| - h_{goal}(\delta))^2$$

where G is the goal of the problem.

For a CNF-state φ, encoded by a set of clauses, the heuristic function of CNF is a combination of the number of satisfied subgoals in φ ($h_{goal}(\varphi)$) and the size of φ, defined as the sum of the sizes of non-unit clauses in φ ($h_{size}(\varphi)$).

System Organization

The proposed systems are organized as in Fig. 1. The first component is a front-end, that acts as a *static analyzer*. The static analyzer is in charge of applying several simplifications and optimizations to the input problem specification—initially expressed in PDDL. The simplified specification (expressed either in PDDL or in the action language $\mathcal{AL}$—the native input format of CPA, DNF, and CNF) produced by the static analyzer is then fed to the actual planner. The separation of the two stages allows us to investigate the use of different planners applied to the same simplified problem specification.

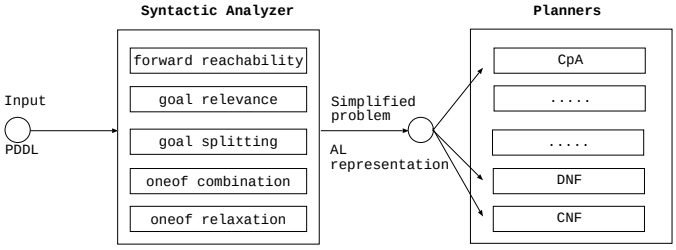

Figure 1: Overall System

The implementation of the static analyzer makes use of the PDDL parser originally developed for these systems; the parser has been modified to enable the construction of a Prolog representation of the problem specification. This Prolog representation is used as the input to the static analyzer, implemented in Prolog. The analyzer implements the basic simplifications, the `oneof`-combination/relaxation, and the goal-splitting algorithm. Its output is a sequence of simplified problems in $\mathcal{AL}$, which serve as input to these planners. An option is also available to produce PDDL output from the static analyzer—that can be fed, for example, to a different planner.

CPA makes use of h_{cs} in combination with a best-first search algorithm. CPA employs an explicit representation of cs-states as sets of sets of propositions, and they make use of the C++ standard library `std` for sets manipulation. To reduce the space consumption, a partial state is created only once and it is shared by all cs-states containing it.

Discussion and Conclusion

We presented the main techniques implemented in the three conformant planners CPA, DNF, and CNF. Experimentally, these planners are competitive with state-of-the-art conformant planners in several benchmark domains. CNF scales better than others in some benchmarks but the overhead in computing the successor belief state slows it down in small instances.

The development of these planners highlights the fact that the representation of belief states can significantly impact the performance of a planner. Some simplification techniques are applicable to a wide range of representations, while others are specific to certain representations. For example, `oneof`-combination is better for DNF-representation and the `oneof`-relaxation is better for CNF-representation.

So far, the proposed planners do focus only on the size of the initial belief state. For scalability, the problems related to the number of actions in the planning problems will need to be addressed as well. We believe that heuristics should provide a solution to this problem and this will be a focus of our future work. Furthermore, our study reveals that there seems to be no "one size fits all" representation for all planning domains. As such, identifying the most useful representation given a planning problem will be another interesting work that we plan to explore in the near future.

References

Baral, C.; Kreinovich, V.; and Trejo, R. 2000. Computational complexity of planning and approximate planning in the presence of incompleteness. *AIJ* 122:241–267.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic search + symbolic model checking = efficient conformant planning. In *IJCAI*, 467–472. Morgan Kaufmann.

Bryce, D., and Kambhampati, S. 2004. Heuristic Guidance Measures for Conformant Planning. In *ICAPS 2004*, 365–375. AAAI.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res.* 22:215–278.

Son, T. C., and Baral, C. 2001. Formalizing sensing actions - a transition function based approach. *AIJ* 125(1-2):19–91.

Son, T. C., and Tu, P. H. 2006. On the Completeness of Approximation Based Reasoning and Planning in Action Theories with Incomplete Information. In *KR*, 481–491.

Son, T. C.; Tu, P. H.; Gelfond, M.; and Morales, R. 2005. Conformant Planning for Domains with Constraints — A New Approach. In *AAAI*, 1211–1216.

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

To, S. T.; Son, T. C.; and Pontelli, E. 2010. A New Approach to Conformant Planning using CNF. In *ICAPS*. To Appear.