Research Workshop of the
Israel Science Foundation

Proceedings of the Workshop on

Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-15)

Edited By:

Roman Barták, Miguel A. Salido

Jerusalem, Israel 7/6/2015

Organizing Committee

Roman Barták
Charles University in Prague, Czech Republic

Miguel A. Salido
Universidad Politécnica de Valencia, Spain

Program committee

Federico Barber, Universidad Politecnica de Valencia, Spain
Roman Barták, Charles University, The Czech Republic
Amedeo Cesta, ISTC-CNR, Italy
Minh Binh Do, NASA Ames Research Center, USA
Agostino Dovier, Universita Degli Studi di Udine, Italy
Enrico Giunchiglia, Universita di Genova, Italy
Weidong Li, Coventry University, UK
Eva Onaindia, Universidad Politecnica de Valencia, Spain
Nicola Policella, European Space Agency, Germany
Hana Rudová, Masaryk University, The Czech Republic
Miguel A. Salido, Universidad Politecnica Valencia, Spain
Torsten Schaub, University of Potsdam, Germany
Dunbing Tang, Nanjing University of Aeronautics&Astronomics, China
Enrico Pontelli, New Mexico State University, USA
Ramiro Varela, Universidad de Oviedo, Spain
Gerard Verfaillie, ONERA, Centre de Toulouse, France
Vincent Vidal, CRIL-IUT, France
Petr Vilím, ILOG, France
Neil Yorke-Smith, American University of Beirut, Lebanon
Neng-Fa Zhou, The City University of New York, USA

Foreword

The areas of AI planning and scheduling have seen important advances thanks to the application of constraint satisfaction models and techniques. Especially solutions to many real-world problems need to integrate plan synthesis capabilities with resource allocation, which can be efficiently managed by using constraint satisfaction techniques.

The workshop aims at providing a forum for researchers in the field of Artificial Intelligence to discuss novel issues on planning, scheduling, constraint programming/constraint satisfaction problems (CSPs) and many other common areas that exist among them. On the whole, the workshop mainly focuses on managing complex problems where planning, scheduling and constraint satisfaction must be combined and/or interrelated, which entails an enormous potential for practical applications and future research. Formulations of P&S problems as CSPs, resource and temporal global constraints, and inference techniques are of particular interest of COPLAS.

This workshop has been hold form 2006 in the context of ICAPS and CP conferences, which provided a broader audience and gave the participants of both events the opportunity to exchange ideas and approaches that lead to a valuable and fruitful discussion, and inspired forthcoming research. COPLAS is ranked as CORE B in ERA Conference Ranking and it is covered in selected Elsevier database products.

Roman Barták, Miguel A. Salido
COPLAS 2015 Organizers
June 2015

Table of Contents

The RANTANPLAN Planner: System Description

Miquel Bofill and **Joan Espasa** and **Mateu Villaret**
Departament d'Informàtica, Matemàtica Aplicada i Estadística
Universitat de Girona, Spain

Abstract

RANTANPLAN is a numeric planning solver that takes advantage of recent advances in SMT. It extends reduction to SAT approaches with an easy and efficient handling of numeric fluents using background theories. In this paper we describe the design choices and features of RANTANPLAN, especially, how numeric reasoning is integrated in the system. We also provide experimental results showing that RANTANPLAN is competitive with existing exact numeric planners.

Introduction

The problem of planning, in its most basic form, consists in finding a sequence of actions that allow to reach a goal state from a given initial state. Although initially considered a deduction problem, it was rapidly seen that it could be addressed by looking at it as a satisfiability (model finding) problem (Kautz and Selman 1992). Many (incomplete) heuristic methods can be found in the literature to efficiently deal with this problem, most of them oriented towards finding models. Exact methods were ruled out at the beginning due to their inefficiency. However, in (Kautz, McAllester, and Selman 1996) it was shown that modern off-the-shelf SAT solvers could be effectively used to solve planning problems. In recent years, the power of SAT technology has been leveraged to planning (Rintanen 2012), making reduction into SAT competitive with heuristic search methods.

Although a lot of work has been devoted to the encoding of plans in propositional logic, only a few works can be found in the literature on satisfiability based approaches to planning in domains that require numeric reasoning. This is probably due to the difficulty of efficiently handling at the same time numeric constraints and propositional formulas. Among the few works dealing with planning with resources are (Hoffmann 2003; Kautz and Walser 1999; Gerevini, Saetti, and Serina 2008; Hoffmann et al. 2007). There have also been some works using constraint and logic programming (Dovier, Formisano, and Pontelli 2010; Barták and Toropila 2010). However, the advances in satisfiability modulo theories (SMT) (Barrett et al. 2009) in the last years make worth considering this alternative. With RANTANPLAN we demonstrate that with SMT one can elegantly handle numeric reasoning inside any PDDL domain, thanks to the integration of various background theories with a SAT solver.

As the number of variables, and hence the search space, rapidly grows with the number of time steps considered, a key idea to improve the performance of SAT-based planners is to consider the possibility of executing several actions at the same time, i.e., the notion of parallel plans. Parallel plans increase the efficiency not only because they allow to reduce the time horizon, but also because it is unnecessary to consider all total orderings of the actions that are performed in parallel. Nevertheless, in SAT-based planning, parallel plans are not intended to represent true parallelism in time, and it is usually required that a sequential plan can be built from a parallel plan in polynomial time. Two main types of parallel plans are considered: $\forall$-step plans, and $\exists$-step plans. In $\forall$-step plans, any ordering of parallel actions must result in a valid sequential plan. In $\exists$-step plans, there must exist a total ordering of parallel actions resulting in a valid sequential plan. We refer the reader to (Rintanen 2009; Rintanen, Heljanko, and Niemelä 2006) for further details. RANTANPLAN supports $\forall$ and $\exists$-step plans, using various different encodings.

To ensure that a parallel plan is sound, it is necessary that all actions proposed to be executed at the same time do not interfere. Different notions of interference have been defined, some more restrictive, some more relaxed. But, as far as we know, for efficiency reasons, potential interference between action is always determined statically, i.e., independently of any concrete state, hence in a fairly restrictive way. Moreover, very few works deal with the notion of incompatibility of actions in planning with resources, most of them with rather syntactic or limited semantic approaches (Kautz and Walser 1999; Fox and Long 2003; Gerevini, Saetti, and Serina 2008) RANTANPLAN incorporates a novel method for determining interference between actions at compile time, using an SMT solver as an oracle.

Summing up, RANTANPLAN is a numeric planner based on planning as satisfiability, which translates PDDL problems into SMT formulas. It supports various types of parallelism, using a novel notion of interference. Experimental results show that it is competitive with other exact numeric planners and strictly better in non-trivial numeric domains.

Related Work

The pioneering work of LPSAT (Wolfman and Weld 1999) on planning with resources can indeed be considered one of

the precursors of SMT, as the basic ideas of SMT (Boolean abstraction, interaction of a SAT solver with a theory solver, etc.) were already present in it.

A comparison between SAT and SMT based encodings for planning in numeric domains can be found in (Hoffmann et al. 2007). In the SAT approach, the possible values of numeric state variables is approximated, by generating a set of values $D_t(v)$ for every numeric variable v, so that every value that v can have after t time steps is contained in $D_t(v)$. These finite domains then serve as the basis for a fully Boolean encoding, where atoms represent numeric variables taking on particular values. With respect to SMT, where numeric variables and expressions are first class citizens, the authors argue that the expressivity of the SMT language comes at the price of requiring much more complex solvers than for SAT and, for this reason, their SAT-based method is very efficient in domains with tightly constrained resources, where the number of distinct values that a numeric variable can take is small.

Other approaches, related to SMT to some amount as well, have been developed more recently. In (Belouaer and Maris 2012), a set of encoding rules is defined for spatio-temporal planning, taking SMT as the target formalism. On the other hand, in (Gregory et al. 2012) a modular framework, inspired in the architecture of lazy SMT, is developed for planning with resources. We compare with some of these approaches in the experimental evaluation section.

Preliminaries

A numeric planning problem is defined as a tuple $\langle V, P, A, I, G\rangle$ where V is a set of numeric variables, P is a set of propositions (or Boolean variables), A is a set of actions, I is the initial state and G is a formula over $V \cup P$ that any goal state must satisfy.

A state is a total assignment to the variables. Actions are formalized as pairs $\langle p, e\rangle$, where p are the preconditions and e the effects. More formally, p is a set of Boolean expressions over $V \cup P$, while e is a set of (conditional) effects of the form $f \Rightarrow d$, where f is a Boolean expression over $V \cup P$ and d is a set of assignments. An assignment is a pair $\langle v, exp\rangle$, where v is a variable and exp is an expression of the corresponding type. For example, increasing a variable v by one is represented by the pair $\langle v, v+1\rangle$, indicating that $v+1$ is the value that v will hold in the next state. Unconditional effects are represented by setting $f = true$.

The active effects of an action $a = \langle p, e\rangle$ in a state s are $\cup_{f \Rightarrow d \in e}\{d \mid s \models f\}$. An action $a = \langle p, e\rangle$ is executable in a given state s if $s \models p$ and the active effects of a in state s are consistent, i.e., we do not have $exp \neq exp'$ for any variable $v \in V \cup P$ and assignments $\langle v, exp\rangle$ and $\langle v, exp'\rangle$ in the active effects.

The state resulting from executing action a in state s is denoted by $apply(a, s) = s'$. The new state s' is defined by assigning new values to the variables according to the active effects, and retaining the values of the variables that are not assigned values by any of the active effects.

A sequential plan of length n for a given planning problem $\langle V, P, A, I, G\rangle$ is a sequence of actions $a_1; a_2; \ldots; a_n$ such that $apply(a_n \ldots apply(a_2, apply(a_1, I)) \ldots) \models G$.

A parallel plan of length n can be defined similarly to a sequential plan. Instead of having a sequence of actions, we have a sequence of sets of actions $\sigma_1; \sigma_2; \ldots; \sigma_n$ such that $order(\sigma_1) \oplus order(\sigma_2) \oplus \cdots \oplus order(\sigma_n)$ is a sequential plan, where $order(\sigma_i)$ is an ordering function which transforms the set σ_i into a sequence of actions, and $\oplus$ denotes the concatenation of sequences. Actions in the same set σ_i are said to occur in parallel.

The notion of parallelism of a $\forall$-step plan is defined as the possibility of ordering the actions of each set to any total ordering, i.e., no two actions a, a' in each σ_i are interfering (e.g., executing a neither falsifies the precondition of a' nor changes any of its active effects, and vice versa).

The $\exists$-step semantics weakens the $\forall$-step requirements, by only requiring the existence of some correct ordering of the actions that results in a valid sequential plan.

In the planning as SAT approach, a planning problem is solved by considering a sequence of formulas ϕ_0, ϕ_1, ϕ_2, ..., where ϕ_i encodes the feasibility of a plan that allows to reach a goal state from the initial state in i steps. The solving procedure proceeds by testing the satisfiability of ϕ_0, ϕ_1, ϕ_2, and so on, until a satisfiable formula ϕ_n is found. It is a matter of the encoding whether one or various (non interfering) actions are executed at each step.

Framework

RANTANPLAN supports a fragment of PDDL which is close to general numeric PDDL 2.1, excluding the temporal extensions and metric optimizations. In terms of requirements, we consider typing, numeric and object fluents, equality as built-in predicate, universally quantified and negative preconditions, and conditional effects.

With respect to numeric effects, we consider $assign(x, exp)$, $increase(x, exp)$ and $decrease(x, exp)$, where exp is any closed formula over linear integer (or real) arithmetic. With respect to preconditions and conditions of numeric effects, we assume that the restrictions imposed on numeric fluents take the same form as exp.

System Architecture

The structure of the RANTANPLAN system is represented in Figure 1. It receives a PDDL instance and domain and parses it. A preprocessing step is then applied, where PDDL's `forall` constructs are expanded, static functors are identified, and precomputable substitutions and arithmetic operations are carried out.

To encode the formulas ϕ_0, ϕ_1, ϕ_2, ..., one of the two encodings described in the following sections (QF_LIA or QF_UFLIA) is carried out, transforming the PDDL problem to a pure SMT problem. Then the problem is iteratively solved, using the chosen SMT Solver as a black box.

A key aspect of the planner is the detection of interferences between parallel actions at compile time, by means of calls to a SMT Solver. In case the user demands a parallel plan, a disabling graph is computed. By *disabling graph* we refer to a directed graph, where nodes are the grounded actions from the planning problem and an edge exists from action a to action a' if the execution of a can affect a' (forbid

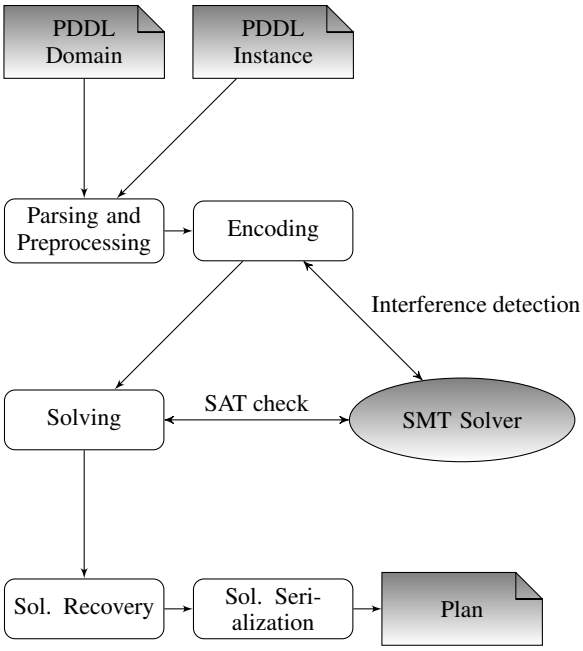

Figure 1: Basic architecture and solving process of the RANTANPLAN solver

its execution or change its active effects) (Rintanen, Heljanko, and Niemelä 2006). This graph is used, depending on the notion of parallelism chosen, to encode the necessary constraints restricting which actions can be carried out at the same time step. In particular, the solver supports:

- A sequential encoding, achieved by using an *at least one* and an *at most one* constraint on the possible actions.
- The $\forall$-step semantics, using the quadratic encoding in (Rintanen, Heljanko, and Niemelä 2006)
- The $\exists$-step semantics, with two encodings. Using the quadratic encoding in (Rintanen, Heljanko, and Niemelä 2006), and a linear-size encoding based on a fixed ordering of operators, also in (Rintanen, Heljanko, and Niemelä 2006).

The system supports solving via API or plain text file using the Yices SMT solver v1.0.38 and the Z3 4.3.2 SMT solver. Once a solution has been found, then it is finally retrieved and serialized. In the following subsections, the relevant aspects of the RANTANPLAN solver are explained in more detail.

QF_LIA Encoding

Numeric planning problems with linear integer arithmetic expressions naturally fall into the QF_LIA logic. In the SMT-LIB standard (Barrett, Stump, and Tinelli 2010), QF_LIA stands for the logic of Quantifier-Free Boolean formulas, with Linear Integer Arithmetic constraints. This logic has a good compromise between expressivity and performance, and is the natural choice for this problem.

We generalized Rintanen's (Rintanen 2009) encoding of planning as SAT to include numeric variables as follows.

For each time step, every ground instance of a PDDL predicate and action is mapped to a Boolean variable, and every ground instance of a PDDL function is mapped to an integer variable. For instance, a predicate stating the position of an aircraft such as `at(?a - aircraft, ?c - city)`, with three cities `c1`, `c2` and `c3`, and two planes `p1` and `p2`, will result into six ground instances `at(p1,c1)`, ..., `at(p2,c3)`, that will be mapped to six Boolean variables $at^t_{p1,c1}$, ..., $at^t_{p2,c3}$ for each time step t. Following the same example, being `at(?o - aircraft) - city` an object fluent, the mapping would result into two integer variables at_{p1}, at_{p2} with the domains being the possible cities `c1`, `c2` and `c3` (these are internally mapped into three distinct integers). Note that thanks to the SMT language, we can get a more compact encoding of states in the presence of object fluents than using a plain SAT approach. The Boolean variables resulting from actions will be used to denote what action is executed at each time step, and with which parameters. The Boolean and integer variables resulting from grounding the predicates and functions, respectively, will constitute the state variables. A superscript t is used to differentiate the variables at each time step.

Given a formula ϕ, by ϕ^t we denote the same formula ϕ where all integer variables x have been replaced by x^t. For the case of assignments, we define:

$$\begin{aligned}
\langle x, true\rangle^t &\stackrel{def}{=} x^t \\
\langle x, false\rangle^t &\stackrel{def}{=} \neg x^t \\
\langle x, k\rangle^t &\stackrel{def}{=} (x^t = k) \\
\langle x, x+k\rangle^t &\stackrel{def}{=} (x^t = x^{t-1} + k) \\
\langle x, x-k\rangle^t &\stackrel{def}{=} (x^t = x^{t-1} - k)
\end{aligned}$$

For each ground[1] action $a = \langle p, e\rangle$, we have the following constraints. First, its execution during time step t implies that its precondition is met:

$$a^t \rightarrow p^t \qquad \forall a = \langle p, e\rangle \in A \tag{1}$$

Also, each of its conditional effects will hold at the next time step if the corresponding condition holds:

$$(a^t \wedge f^t) \rightarrow d^{t+1} \quad \forall a = \langle p, e\rangle \in A, \forall f \Rightarrow d \in e \tag{2}$$

Here we view sets d of literals as conjunctions of literals. Recall also that unconditional effects will have $true$ as condition f.

Second, we need explanatory axioms to express the reason of a change in state variables. For each variable x in $V \cup P$:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{a=\langle p,e\rangle \in A} \left(a^t \wedge (EPC_x(a))^t\right) \tag{3}$$

[1] By a ground action $\langle p, e\rangle$ we refer to an action where p and e are built on the state variables that result from grounding a PDDL model, as explained above.

where, given an action $a = \langle p, e \rangle$ and a variable x,

$$EPC_x(a) = \bigvee_{f \Rightarrow d \in e} \{f \mid d \text{ contains an assignment for } x\}$$

that is, the *effect precondition* for the modification of x in action a, where the empty disjunction is defined as $false$. For Boolean variables, the expression $x^t \neq x^{t+1}$ can be written as $(x^t \wedge \neg x^{t+1}) \vee (\neg x^t \wedge x^{t+1})$.

Interference Between Actions

As said in the introduction, a key concept in parallel plans is the notion of interference between actions. This issue has been carefully considered by Rintanen et al. (Rintanen, Heljanko, and Niemelä 2006) in the setting of planning as SAT. Given a disabling graph, where an edge exists from action a to action a' if the execution of a can affect a', we know for example that the simultaneous execution of all actions pertaining to a strongly connected component is not possible, as given all possible orderings of actions, all of them contain a cycle (and thus they cannot be serialized).

Note that acyclicity is a sufficient but not necessary condition for a set of actions to be executable in some order, since disabling graphs are computed independently of any state.

In (Rintanen 2009), an action a_1 is defined to *affect* another action a_2 if a_1 may prevent the execution of a_2 or change its active effects, and two actions a_1 and a_2 are considered to *interfere* if a_1 affects a_2 or a_2 affects a_1. In $\forall$-step plans, where all possible serializations must be valid, no two interfering actions can occur in parallel. In the more relaxed notion of parallelism of $\exists$-step plans, where it is only required that no action affects a later one in some total ordering, often much more parallelism is allowed in practice. For efficiency reasons, typically syntactic (rather than semantic) restrictions are imposed on parallel actions. For example, in (Rintanen 2009), where only Boolean variables are considered, $a_1 = \langle p_1, e_1 \rangle$ is determined to affect $a_2 = \langle p_2, e_2 \rangle$ if, for some variable a,

1. a is set to $true$ in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and a occurs negatively in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$, or
2. a is set to $false$ in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and a occurs positively in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$.

That is, a_1 affects a_2 if a_1 can impede the execution of a_2, or change its effects. Note that this is not a symmetric relation.

This is a fully syntactic check which can be used to establish sufficient although not necessary conditions for finding serializable parallel plans. We can observe that interference between effects is not considered. This is because, in the case two actions have contradictory effects, any formula encoding a plan with those two actions running in parallel will become unsatisfiable.

The previous approach could be naively generalized to the case of numeric variables as follows: an action $a_1 = \langle p_1, e_1 \rangle$ *affects* an action $a_2 = \langle p_2, e_2 \rangle$ if, for some variable x, x is modified in d_1 for some $f_1 \Rightarrow d_1 \in e_1$, and x occurs in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in e_2$.

Performing only syntactic checks like the previous seems too much restrictive for numeric variables, even in the case that we determine interference at compile time, i.e., independently of any concrete state. For this reason, we propose a new idea, currently submitted for review (Bofill, Espasa, and Villaret 2015), which is to use SMT technology to perform semantic checks of interference at compile time, in order to increase the amount of parallelization of numeric plans.

Our method is independent of any test suite and does not require any special purpose algorithm, as it relies on encoding the possible interference situations between pairs of actions as SMT formulas and checking their satisfiability, by calling an SMT solver, at compile time. For example, an important difference with the purely syntactic definition of interference of (Rintanen 2009) is that we include the preconditions of the actions in the semantic checks. More precisely, two actions can occur in parallel only if their preconditions can be satisfied simultaneously, regardless of the variables they contain. This way, we are able to avoid many "false positive" interference relationships.

All in all, we obtain a much more fine-grained notion of interference, that we expect will help to increase the parallelization of actions. Note that the interference relationships determined semantically will always be a subset of the interference relationships determined syntactically. Interestingly, we will be using an SMT solver both at compile time, as an oracle to predict interference relationships, and at solving time.

For efficiency reasons, to perform the interference checks we do not consider grounded actions, but the original actions in the PDDL model. Since now actions are not instantiated, we need to unify the parameters of the same type in the actions for which we check interference.

Imagine we have two actions, say `move(?d - ship ?a ?b - location)` and `dock(?e - ship ?c - location)`. We will be interested to know, for example, if the actions interfere in the case that `?d` and `?e` are the same ship. Or in the case that locations `?a` and `?c` are the same, etc. In conclusion, we must consider all different possible equality and disequality relationships between parameters of the same type, to find out in which cases one action can interfere with another action.

To accomplish this task we group all the parameters of the two considered actions by its most general declared type. Following the previous example, in total we have three parameters `a`, `b` and `c` of the type `location` and two parameters `d` and `e` of the type `ship`. Therefore, we need to check interference in the following situations:

```
a = b = c, d = e
a = b, b != c, d = e
a = c, b != c, d = e
a != b, b = c, d = e
...
```

Internally the parameters are substituted by equal or different integers according to the generated constraints, and the formulas encoding incompatibility are checked for satisfiability. These consistency checks can be done in a reasonable time with an SMT solver, and the amount of par-

allelism achieved is significantly higher than with syntactic approaches. To illustrate the situations where this notion of interference is especially accurate, consider the following example. The problem consists in transporting people between cities using planes. Each plane has a limited number of seats and a given fuel capacity. The actions on this domain are `fly`, `board`, `debark` and `refuel`. We focus on the `fly` and `board` actions. A plane can only fly if it is transporting somebody and it has enough fuel to reach its destination, and boarding is limited by seat availability:

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)

 :precondition (and (at ?a ?c1)
                (> (onboard ?a) 0)
                (>= (fuel ?a)
                    (distance ?c1 ?c2)))

 :effect (and (not (at ?a ?c1))
          (at ?a ?c2)
          (decrease (fuel ?a)
                    (distance ?c1 ?c2))))

(:action board
 :parameters (?p - person
              ?a - aircraft
              ?c - city)

 :precondition (and (at ?p ?c)
                (at ?a ?c)
                (> (seats ?a) (onboard ?a)))

 :effect (and (not (at ?p ?c))
          (in ?p ?a)
          (increase (onboard ?a) 1)))
```

The syntactic notion of interference would determine interference between `fly` and `board`, since `board` modifies the `onboard` function (number of passengers) and `fly` checks the value of this function in its precondition. On the contrary, with the semantic technique, we would find out that there is no interference at all, since it is impossible that the preconditions of `board` and `fly` were true at the same time, and after executing `board` the precondition of `fly` became false. Note that the precondition of `fly` requires `(> (onboard ?a) 0)` and the effect `(increase (onboard ?a) 1)` of `board` can never falsify `(> (onboard ?a) 0)`.

Sequential Plans

The sequential encoding allows exactly one action per time step. This is achieved by imposing an *exactly one* constraint on the action variables at each time step. We tested some well-known encodings, and we settled with the binary encoding (Frisch and Giannaros 2010) as it gave us the best performance. This encoding introduces new variables $B_1, \ldots, B_{\lceil \log_2 n \rceil}$, where $n = |A|$, and associates each variable a_i^t with a unique bit string $s_i \in \{0,1\}^{\lceil \log_2 n \rceil}$. The encoding is:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg a_i^t \vee \odot(i,j) \tag{4}$$

$$\bigvee_{i=1}^{n} a_i^t \tag{5}$$

where $\odot(i,j)$ is B_j if the j^{th} bit of the bit string of s_i is 1, and $\neg B_j$ otherwise. The binary encoding of the *at most one* constraint (4), introduces $\lceil \log_2 n \rceil$ new variables and $n\lceil \log_2 n \rceil$ binary clauses. Together with the *at least one* constraint (5), we obtain the desired *exactly one* constraint.

Parallel Plans

Encodings for two types of parallel plan semantics have been implemented in RANTANPLAN: $\forall$-step plans, and $\exists$-step plans.

$\forall$-step Plans The notion of parallelism of a $\forall$-step plan is defined as the possibility of ordering the actions of each time step to any total order. Therefore, at each time step t we simply add a mutex between any pair of interfering actions a_i and a_j:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ or } a_j \text{ affects } a_i \tag{6}$$

$\exists$-step Plans In $\exists$-step plans, there must exist at least a total ordering of parallel actions resulting in a valid sequential plan. RANTANPLAN implements a quadratic encoding for this purpose. It takes as ingredient an arbitrary total ordering $<$ on the actions, and the parallel execution of two actions a_i and a_j such that a_i affects a_j is forbidden only if $i < j$:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ and } i < j \tag{7}$$

The linear-size encoding described in (Rintanen, Heljanko, and Niemelä 2006), is also supported.

Since $\exists$-step plans are less restrictive than $\forall$-step plans, as they do not require that all orderings of parallel actions result in valid sequential plan, they normally allow more parallelism.

Plan Serialization

To obtain a sequential plan from the solution, for each time step with more than one action, a subgraph of the disabling graph is extracted, containing only the actions at that time step. A valid order between actions can then be computed.

Since in all implemented parallel encodings acyclicity is guaranteed between the executed actions, a reversed topological order of the subgraph is always as a valid order.

Extension: QF_UFLIA Encoding

As the previously introduced QF_LIA encodings grows considerably with the time horizon, to the point of getting unmanageable instances, we have started to develop a more compact encoding, using the theory of uninterpreted functions to express predicates, functions and actions. This encoding is reminiscent of the lifted causal encodings in (Kautz, McAllester, and Selman 1996).

In the SMT-LIB standard (Barrett, Stump, and Tinelli 2010), QF_UFLIA stands for the logic of Quantifier-Free

Boolean formulas, with Linear Integer Arithmetic constraints and Uninterpreted Functions. Uninterpreted functions have no other property than its name and arity, and are only subject to the following axiom: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \rightarrow f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$.

The encoding goes as follows. Every defined object in the problem is mapped to an integer. For each function, predicate and action, an uninterpreted function is declared, with each parameter being declared as an integer. Also, a new integer parameter is added to each of them, representing a time step. Uninterpreted functions corresponding to predicates and actions return a Boolean value, whilst the ones for functions return an integer value. Moreover, for each action, parameter and time step, a new integer variable is defined, representing the value of that parameter in the action if executed at the corresponding time step.

For example, the Boolean function $\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t)$ determines whether action a with parameters $x^t_{a,1}, \ldots, x^t_{a,n}$ is executed at time step t. The parameter t is a constant, which is shared between all uninterpreted functions for the actions, predicates and functions in the same time step. Contrarily, $x^t_{a,1}, \ldots, x^t_{a,n}$ are variables with finite domains, and constraints are added to restrict their possible values. Regarding predicates and functions, no new variables are defined, since their arguments will be either constants or variables occurring in some action.

We remark that, in this new setting, a state is defined by the value of the uninterpreted functions corresponding to predicates and functions, for a given value of their arguments. Equations (1) and (2) of the QF_LIA encoding are generalized here as:

$$\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t) \rightarrow p^t \quad \forall a = \langle p, e \rangle \in A \qquad (8)$$

$$\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t) \wedge f^t \rightarrow d^{t+1} \quad \forall a = \langle p, e \rangle \in A, \forall \langle f, d \rangle \in e \qquad (9)$$

Note that this results in a much more compact encoding than if we restrict to QF_LIA, since here we are using variables as arguments of functions, and it is the SMT solver who is in charge of guessing the concrete values of the parameters of the executed actions. The considered set of actions A is now parametrized, and hence similar to that of PDDL, with actions like $\mathit{fly}(x, y, z)$, instead of grounded actions like $\mathit{fly}_{p1,c1,c1}$, $\mathit{fly}_{p1,c1,c2}$, etc. Equation (3) is generalized as:

$$\varphi_h(c_{h,1}, \ldots, c_{h,n}, t) \neq \varphi_g(c_{h,1}, \ldots, c_{h,n}, t+1) \rightarrow \bigvee_{a \in touch(g)} \Big(\varphi_a(x^t_{a,1}, \ldots, x^t_{a,m}, t) \bigwedge_{\substack{i \in 1..n, j \in 1..m \\ name(h,i) = name(a,j)}} (x^t_{a,j} = c_{h,i}) \Big)$$
$$\forall h \in H, \forall c_{h,1}, \ldots, c_{h,n} \in S_1 \times \cdots \times S_n \qquad (10)$$

where H is the set of predicates and functions, $touch(h)$ is the set of actions that may modify h, S_i is the domain of the i-th argument of φ_h, and $name(h, k)$ is the name in the PDDL model of the k-th argument of the functor h. To help the reader understand the formula, we provide an example. Suppose we have the following simple PDDL problem:

- objects: `A,B - truck, L1,L2,L3 - loc`
- predicate: `at(?t - truck, ?l - loc)`
- actions:
  `travel(?t - truck, ?from - loc, ?to - loc)`
  `refuel(?x - truck, ?where - loc)`
- function: `fuel(?t - truck) - number`

where `travel` has `(decrease (fuel ?t) 10)` among its effects, and `refuel` has `(increase (fuel ?x) 20)` as its only effect. Constraint (10) for the `fuel` function would be encoded into SMT at time step 0 as follows:

```
(=> (distinct (fuel A 0) (fuel A 1))
  (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 A))
      (and (refuel x4_0 x5_0 0) (= x4_0 A))))

(=> (distinct (fuel B 0) (fuel B 1))
  (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 B))
      (and (refuel x4_0 x5_0 0) (= x4_0 B))))
```

That is, we are saying that if the fuel of truck `A` (or `B`) has changed this should be because it has been the protagonist of some action implying a modification in its fuel, namely traveling or refueling.

Again, this is much more compact than its QF_LIA counterpart. With respect to the parallelism, for now this encoding only supports the sequential plan semantics, as encoding parallelism using this encoding is not straightforward. This approach is currently under development, as we obtained encouraging preliminary experimental results (Bofill, Espasa, and Villaret 2014).

Experimental Evaluation

In this section we report on experiments with RANTANPLAN using Yices (Dutertre and De Moura 2006) v1.0.38 as backend solver. All experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz.

The goal of the experiments is to evaluate if RANTANPLAN is competitive with state of the art exact numeric planners, as well as showing the benefits of having a good notion of interference.

For the sake of simplicity, only QF_LIA $\exists$-step plans are considered, using a quadratic encoding for expressing incompatibility of actions. We experimentally observed that the solver behavior was more stable when using a quadratic encoding than when using a linear encoding, probably because a linear encoding is more perturbable with the chosen ordering of actions.

We consider four distinct domains: the numeric versions of *ZenoTravel* and *Depots*, the real-life challenging *Petrobras* domain, and the crafted *Planes* domain, shown in Figure 2. All instances have been translated to make use of

```
(define (domain planes)
(:requirements :typing :fluents)
(:types  city locatable - object
         aircraft person - locatable)
(:functions
 (at       ?x - locatable) - city
 (in       ?p - person) - aircraft
 (fuel     ?a - aircraft) - number
 (seats    ?a - aircraft) - number
 (capacity ?a - aircraft) - number
 (onboard  ?a - aircraft) - number
 (distance ?c1 - city ?c2 - city) - number)

(:action board
 :parameters (?p - person
              ?a - aircraft
              ?c - city)
 :precondition (and (= (at ?p) ?c)
                (= (at ?a) ?c)
                (> (seats ?a) (onboard ?a)))
 :effect (and (assign (at ?p) undefined)
          (assign (in ?p) ?a)
          (increase (onboard ?a) 1)))

(:action debark
 :parameters (?p - person
              ?a - aircraft
              ?c - city)
 :precondition (and (= (in ?p) ?a)
                (= (at ?a) ?c))
 :effect (and (assign (in ?p) undefined)
          (assign (at ?p) ?c)
          (decrease (onboard ?a) 1)))

(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (= (at ?a) ?c1)
                (> (onboard ?a) 0)
                (>= (fuel ?a)
                    (distance ?c1 ?c2)))
 :effect (and (assign (at ?a) ?c2)
          (decrease (fuel ?a)
                    (distance ?c1 ?c2)))
 )

(:action refuel
 :parameters (?a - aircraft)
 :precondition (and
                (< (* (fuel ?a) 2) (capacity ?a))
                (= (onboard ?a) 0))
 :effect (and (assign (fuel ?a) (capacity ?a)))))
```

Figure 2: PDDL model of the Planes domain.

object fluents, in order to obtain a compact representation in the translation to SMT. The *Planes* domain was crafted due to the limited interest of the other domains with respect to numeric interactions between actions. This new domain was derived from *ZenoTravel*, by adding some plausible numeric constraints that will help us demonstrate the goodness of the semantic approach when determining potential interference between actions.

We compare the performance of RANTANPLAN with the exact numeric planner NumReach/SAT (Hoffmann et al. 2007) using MiniSAT 2.2.0, and NumReach/SMT using Yices v1.0.38. For NumReach/SMT, we had to adapt its output so it could be used with modern SMT solvers. Moreover, since NumReach supports neither object fluents nor conditional effects, the models have been properly adapted.

Table 1 shows the results for the domains considered using the RANTANPLAN system. The **Syntactic** column shows

Depots Domain

n	Syntactic	Semantic	Time	Edges
1	4.1 (6)	2.8 (6)	31.4%	41.7%
2	32.0 (9)	18.3 (8)	42.8%	44.2%
3	166.9 (13)	108.9 (13)	34.8%	44.9%
4	438.3 (14)	323.0 (14)	26.3%	45.1%
5	TO (8)	TO (17)	-	45.1%
6	TO (-)	MO (1)	-	-
7	188.1 (10)	131.0 (10)	30.4%	44.0%
8	MO (3)	MO (10)	-	44.5%

Zenotravel Domain

n	Syntactic	Semantic	Time	Edges
1	0.0 (0)	0.0 (0)	35.3%	76.3%
2	0.1 (3)	0.0 (3)	23%	74.3%
3	0.2 (3)	0.1 (3)	34.6%	66%
4	0.3 (4)	0.1 (4)	43.5%	66.2%
5	0.5 (4)	0.3 (4)	38.9%	71.5%
6	0.8 (6)	0.5 (6)	43.5%	72.1%
7	0.8 (5)	0.4 (5)	47%	72.6%
8	2.8 (5)	1.7 (5)	38.5%	68.3%
9	26.5 (8)	31.0 (8)	-16.9%	69.6%
10	41.6 (8)	61.9 (8)	-48.7%	70.7%
11	7.1 (6)	4.5 (6)	37.2%	69%
12	105.8 (7)	95.1 (7)	10.1%	70.4%
13	1288.3 (9)	1291.5 (9)	-0.2%	72.6%
14	TO (7)	TO (7)	-	55.0%

Petrobras Domain

n	Syntactic	Semantic	Time	Edges
1	14.7 (3)	8.8 (3)	40.7%	50.4%
2	19.3 (4)	11.2 (4)	42.2%	51.8%
3	24.6 (5)	14.0 (5)	43.2%	53.2%
4	47.0 (8)	28.2 (8)	40.1%	54.5%
5	74.9 (9)	59.5 (9)	20.5%	55.8%
6	133.9 (10)	108.7 (10)	18.8%	57.1%
7	700.1 (13)	475.1 (13)	32.1%	58.3%
8	833.4 (13)	800.0 (13)	4.0%	59.5%

Planes Domain

n	Syntactic	Semantic	Time	Edges
1	1.0 (13)	0.3 (10)	71.5%	84.5%
2	6.0 (16)	1.1 (12)	81.2%	84.5%
3	49.9 (18)	8.3 (13)	83.4%	86.5%
4	431.1 (21)	40.0 (15)	90.7%	86.5%
5	117.2 (20)	27.0 (15)	77.0%	86.1%
6	1294.6 (23)	193.3 (18)	85.1%	86.1%
7	621.9 (21)	70.9 (16)	88.6%	85.8%
8	834.2 (22)	105.7 (17)	87.3%	85.8%
9	TO (23)	2889.1 (20)	-	88.0%

Table 1: Time in seconds followed by the number of parallel steps of the plan found between parentheses, for each instance. TO stands for time out and MO for memory out. Cutoff set to 3600 seconds. The Time and Edges columns show the reduction in time and edges of the disabling graph, respectively, when using the semantic approach. Instances where all approaches timed out are omitted.

the results using the generalization of the interference notion of (Rintanen 2009), described at the beginning of subsection "Interference Between Actions", additionally forbidding any two actions to occur in parallel if they modify the same numeric variable. The **Semantic** column shows the results with the lifted semantic notion of interference. In these two columns the number of parallel steps of the valid plan is found between parentheses. In case of a time out (TO) the number between parentheses is the last plan length considered.

The **Time** column shows how much faster each instance is solved with the semantic notion of interference, and the **Edges** column shows which percentage of edges of the disabling graph can be avoided thanks to this new interference notion. Note that even in instances that need the same amount of time steps, the reduction of edges in the disabling graph affects positively on the solving time. This is probably because we are reducing the number of clauses that do not contribute at all to the problem.

	Accumulated steps		Averaged reductions	
Family	Syntactic	Semantic	Time	Edges
Depots	52	51	33.1%	44.2%
ZenoTravel	68	68	22.0%	70.8%
Petrobras	65	65	30.2%	59.1%
Planes	154	116	84.3%	86.8%

Table 2: Summarized results for the domains considered using the RANTANPLAN system with the syntactic and the semantic notions of interference. For each domain we report the total number of steps of the commonly solved instances, and their averaged reductions in solving time and number of edges of the disabling graph.

Table 2 gives the number of accumulated parallel time steps used to reach a valid plan on the commonly solved instances by the two methods implemented in RANTANPLAN. The other columns show the averaged solving time reduction and disabling graph edge reduction. Note that even in domains that maintain the same number of time steps, the reduced disabling graphs make solving times notably smaller.

Table 3 shows the results for the domains considered, comparing NumReach with the semantic version of RANTANPLAN. NumReach does a good job with the *Depots* and *ZenoTravel* domains, but its performance decreases in more complex numeric domains like *Petrobras* and *Planes*, where the range of possible values for numeric fluents tends to grow.

It can be seen that on the *Planes* domain, containing only a few non-trivial numeric constraints, classical approaches (Syntactic and NumReach) tend to be overly restrictive with respect to incompatibility between actions. In most instances it can be observed an important gap between the number of time steps needed to find a valid plan by NumReach and our semantic approach. This is also generally reflected in terms of solving time.

Table 4 lists the total number of instances of each family, the number of instances solved by NumReach/SAT and NumReach/SMT and the number of instances solved by the presented semantic approach. The last two columns give the

Depots Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.0 (6)	1.5 (6)	2.8 (6)
2	0.5 (9)	8.4 (9)	18.3 (8)
3	5.7 (13)	43.1 (13)	108.9 (13)
4	10.1 (15)	134.7 (15)	323.0 (14)
7	2.5 (11)	35.1 (11)	131.0 (10)
8	TO (-)	362.7 (15)	MO (10)
10	4.8 (11)	101.2 (11)	MO (-)
13	2.9 (10)	96.3 (10)	TO (-)
14	25.1 (16)	1650.0 (13)	TO (-)
16	2.2 (9)	118.8 (9)	TO (-)
17	6.8 (8)	313.2 (8)	TO (-)
19	18.1 (11)	849.4 (11)	TO (-)

Zenotravel Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.0 (2)	0.2 (2)	0.0 (0)
2	0.0 (7)	1.5 (7)	0.0 (3)
3	0.1 (6)	3.7 (6)	0.1 (3)
4	0.0 (6)	2.4 (6)	0.1 (4)
5	0.1 (7)	7.0 (7)	0.3 (4)
6	0.0 (7)	4.2 (7)	0.5 (6)
7	0.1 (8)	9.1 (8)	0.4 (5)
8	0.4 (7)	8.1 (7)	1.7 (5)
9	0.3 (9)	18.1 (9)	31 (8)
10	0.7 (9)	24.2 (9)	61.9 (8)
11	3.5 (8)	18.4 (8)	4.5 (6)
12	3.8 (10)	99.6 (10)	95.1 (7)
13	22.2 (11)	555.6 (11)	1291.5 (9)
14	TO (-)	537.4 (9)	TO (7)

Petrobras Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	0.4 (6)	39.8 (6)	8.8 (3)
2	9.8 (9)	56.4 (6)	11.2 (4)
3	17.8 (10)	93.9 (7)	14.0 (5)
4	118.3 (11)	256.5 (9)	28.2 (8)
5	317.9 (14)	312.3 (9)	59.5 (9)
6	325.4 (14)	277.2 (9)	108.7 (10)
7	TO (-)	818.1 (11)	475.1 (13)
8	TO (-)	2753.6 (12)	800.0 (13)

Planes Domain

n	NumReach/SAT	NumReach/SMT	Semantic
1	TO (-)	36.4 (15)	0.3 (10)
2	3.3 (18)	37.9 (18)	1.1 (12)
3	TO (-)	229.9 (20)	8.3 (13)
4	4.4 (22)	632.0 (23)	40.0 (15)
5	TO (-)	768.4 (22)	27.0 (15)
6	TO (-)	1183.7 (25)	193.3 (18)
7	TO (-)	1241.2 (23)	70.9 (16)
8	5.0 (24)	1278.2 (24)	105.7 (17)
9	TO (-)	TO (-)	2889.1 (20)
12	15.5 (21)	TO (-)	TO (19)

Table 3: Time in seconds followed by the number of parallel steps of the plan found between parentheses, for each instance. TO stands for time out and MO for memory out. Cutoff set to 3600 seconds. Instances where all systems timed out are omitted.

	#	Solved instances			Accum. steps	
		N/SAT	N/SMT	Sem.	N/SMT	Sem.
Dep.	22	11	12	5	54	51
Zen.	20	13	14	13	97	68
Petr.	15	6	8	8	69	65
Plan.	12	4	8	9	170	116

Table 4: Summarized results for the domains considered using NumReach/SAT, NumReach/SMT and RANTANPLAN with the semantic notion of interference. For each domain we report the number of solved instances and their accumulated time steps of the commonly solved ones.

number of accumulated parallel time steps used to reach a valid plan on the commonly solved instances.

Note that the amount of parallelism in RANTANPLAN is notable. With respect to the number of steps, RANTANPLAN is strictly more parallel than NumReach/SAT and NumReach/SMT in nearly all instances.

The only domain where the RANTANPLAN planner is not competitive is the *Depots* domain. It is obvious that the reachability approach of NumReach is more adequate for this domain. Moreover NumReach/SAT dominates NumReach/SMT in this domain. This happens because the numeric reasoning present in the domain is nearly null: the only functions present are for controlling load limits of trucks, and thus this domain is perfectly adequate for the approach used by NumReach/SAT. The use of a Linear Integer Arithmetic solver in the RANTANPLAN planner is overkill and a leaner and more efficient approach should be taken for problems of this kind.

Conclusions and Further Work

We have presented RANTANPLAN, a new system for the setting of exact numeric planning. The planner is based on translation into SMT using a planning as satisfiability approach. It takes advantage of background theories in SMT to easily and transparently handle numeric fluents. Moreover it uses an SMT solver at compile time to detect in advance incompatibility between actions. This incompatibility results from lifting the interference notion of (Rintanen 2009) to the setting of planning with resources. We have argued why the presented approach to interference between actions with numeric fluents is better than purely syntactically based ones, and provided empirical evidence of its usefulness.

We have also shown that our system is competitive with the state of the art exact numeric planner NumReach.

As future work, it should be studied how to incorporate the concepts of relaxed $\exists$-plans (Wehrle and Rintanen 2007; Balyo 2013) to our notions of incompatibility and further develop the more compact QF_UFLIA encoding.

Acknowledgments

All authors supported by the Spanish Ministry of Economy and Competitiveness through the project HeLo (ref. TIN2012-33042). Joan Espasa also supported by UdG grant (BR 2013).

References

Balyo, T. 2013. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, 865–871. IEEE.

Barrett, C.; Sebastiani, R.; Seshia, S.; and Tinelli, C. 2009. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, volume 185. IOS Press. chapter 26, 825–885.

Barrett, C.; Stump, A.; and Tinelli, C. 2010. The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`.

Barták, R., and Toropila, D. 2010. Solving sequential planning problems via constraint satisfaction. *Fundamenta Informaticae* 99(2):125–145.

Belouaer, L., and Maris, F. 2012. SMT Spatio-Temporal Planning. In *ICAPS 2012 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2012)*, 6–15.

Bofill, M.; Espasa, J.; and Villaret, M. 2014. Efficient SMT Encodings for the Petrobras Domain. In *Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation (ModRef 2014)*, 68–84.

Bofill, M.; Espasa, J.; and Villaret, M. 2015. On Interference between Actions in Planning with Resources. (submitted).

Dovier, A.; Formisano, A.; and Pontelli, E. 2010. Multivalued action languages with constraints in CLP (FD). *Theory and Practice of Logic Programming* 10(02):167–235.

Dutertre, B., and De Moura, L. 2006. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International. Available at `http://yices.csl.sri.com`.

Fox, M., and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)* 20:61–124.

Frisch, A. M., and Giannaros, P. A. 2010. SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow. In *10th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2008. An approach to efficient planning with numerical fluents and multicriteria plan quality. *Artificial Intelligence* 172(8):899–944.

Gregory, P.; Long, D.; Fox, M.; and Beck, J. C. 2012. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI.

Hoffmann, J.; Gomes, C. P.; Selman, B.; and Kautz, H. A. 2007. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 1918–1923.

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

Kautz, H., and Selman, B. 1992. Planning as Satisfiability. In *10th European Conference on Artificial Intelligence (ECAI 92)*, 359–363. John Wiley & Sons, Inc.

Kautz, H., and Walser, J. P. 1999. State-space planning by integer optimization. In *AAAI/IAAI*, 526–533.

Kautz, H. A.; McAllester, D. A.; and Selman, B. 1996. Encoding Plans in Propositional Logic. In *Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR 96)*, 374–384. Morgan Kaufmann.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2009. Planning and SAT. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 483–504.

Rintanen, J. 2012. Planning as Satisfiability: Heuristics. *Artificial Intelligence* 193:45–86.

Wehrle, M., and Rintanen, J. 2007. Planning as Satisfiability with Relaxed Exist-Step Plans. In *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence*, volume 4830 of *LNCS*, 244–253. Springer.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT Engine & Its Application to Resource Planning. In *Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 99)*, 310–317. Morgan Kaufmann.

Compiling planning into quantum optimization problems: a comparative study

Bryan O'Gorman, Eleanor G. Rieffel, Minh Do, Davide Venturelli, Jeremy Frank
NASA Ames Research Center
Moffett Field, CA 94035
firstname.lastname@nasa.gov

Abstract

One approach to solving planning problems is to compile them to another problem for which powerful off-the-shelf solvers are available; common targets include SAT, CSP, and MILP. Recently, a novel optimization technique has become available: quantum annealing (QA). QA takes as input problem instances encoded as Quadratic Unconstrained Binary Optimization (QUBO). Early quantum annealers are now available, and more sophisticated quantum annealers will likely be built over the next decades. Specific quantum annealing hardware implementations have specific constraints, restricting the types of QUBOs each can take as input. In this paper, we introduce the planning community to the key steps involved in compiling planning problems to quantum annealing hardware: a hardware-independent step, mapping, and a hardware-dependent step, embedding. After describing two approaches to mapping general planning problems to QUBO, we provide preliminary results from running an early quantum annealer on a parameterized family of hard planning problems. The results show that different mappings can lead to a substantial difference in performance, even when many superficial features of the resulting instances are similar. We also provide some insights gained from this early study, and suggest directions for future work.

Introduction

One approach to solving planning problems is to compile them to another problem for which powerful off-the-shelf solvers are available; common targets include SAT, CSP, and MILP. Recently, a novel optimization technique has become available: quantum annealing. Quantum annealing is one of the most accessible quantum algorithms for a computer science audience not versed in quantum computing because of its close ties to classical optimization algorithms such as simulated annealing.

While large-scale universal quantum computers are likely decades away from realization, we expect to see a variety of special-purpose quantum-computational hardware emerge within the next few years. Already, early quantum annealers are available, and more sophisticated quantum annealers will be built over the next decades. While certain classes of problems are known to be more efficiently solvable on a universal quantum computer (Rieffel and Polak 2011; Nielsen and Chuang 2001), for the vast majority of problems the computational power of quantum computing is unknown. Until quantum hardware became available it was impossible to empirically evaluate heuristic quantum algorithms such as quantum annealing.

While there are intuitive reasons why quantum annealing may be able to outperform classical methods on some classes of optimization problems, the effectiveness of quantum annealing is as yet poorly understood. Our work is the first to explore the use of quantum annealing to attack problems arising in planning and scheduling. This work explores compilation of planning problems to quadratic unconstrained binary optimization (QUBO) problems, the type of problem that quantum annealers are designed for. While the immaturity of the technology means that current results are limited, the significant performance differences that result from different compilation approaches suggest that subtle issues are at play in determining the best compilation approaches for quantum annealers.

In this paper, we introduce the planning community to the key steps involved in compiling planning problems to quantum annealing hardware. Figure 1 shows the main steps in our framework of solving STRIPS planning problems (Fikes and Nilsson 1972; Ghallab, Nau, and Traverso 2004) represented in PDDL using a D-Wave quantum annealer housed at NASA Ames Research Center: **mapping** the problems to QUBO, and **embedding**, which takes these hardware-independent QUBOs to other QUBOs that matches the specific quantum annealing hardware that will be used. While debate continues as to the extent to which the D-Wave machine is quantum (Johnson et al. 2011b; Boixo et al. 2013; Smolin and Smith 2013; Wang et al. 2013; Boixo et al. 2014; Shin et al. 2014b; Vinci et al. 2014; Shin et al. 2014a), these machines provide the first opportunity for researchers to experiment with quantum annealing. This work does not aim to contribute to that debate, but rather examines different mappings of application problems to quantum annealing to give insight into their relative strengths and weaknesses as best we can with current technology.

We describe two approaches to mapping general STRIPS planning problems to QUBO problems. The mappings were described in (Rieffel et al. 2014b), where one is a variant of the mapping described in (Smelyanskiy et al. 2012). We explore the properties of these mappings for a parametrized family of scheduling-type planning problems based on graph coloring (Rieffel et al. 2014a). We discuss preliminary re-

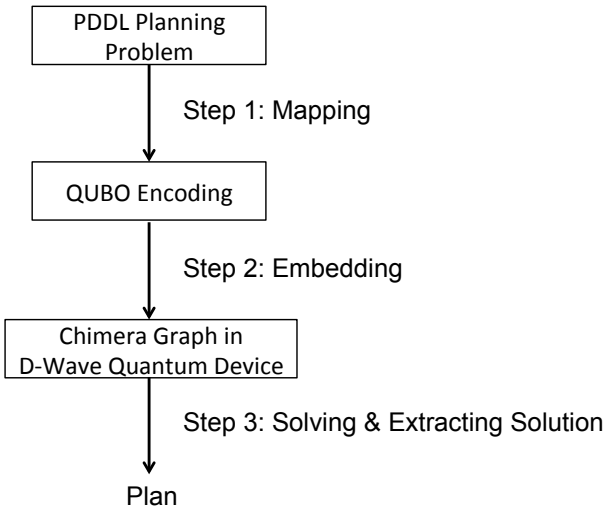

Figure 1: The main steps in our approach of using quantum annealer to solve a planning problem.

sults from an early quantum annealer, a D-Wave Two machine, applied to these problems under the two general mappings. Due to the current hardware limitation of existing quantum annealers, our empirical evaluation has been conducted on a limited set of small planning problems. Nevertheless, these early results show that different compilations to QUBO can lead to substantial differences in performance, even when many features of both the mapping and embedded QUBOs are similar.

Our paper is a reworking and deepening of our paper (Rieffel et al. 2014b) to target planning and scheduling researchers, rather than quantum computing researchers. Our main contributions are:

- A description of quantum annealing with example applications to planning;
- Two different ways of compiling general planning problems to QUBO; and
- Results from runs of a parametrized family of scheduling-type planning problems on an early quantum annealer.

We begin with an overview of quantum annealing, including the mapping and embedding compilation steps.

Ingredients of Quantum Annealing

Quantum annealing (Farhi et al. 2000; Das and Chakrabarti 2008; Johnson et al. 2011a; Smelyanskiy et al. 2012) is a metaheuristic for solving optimization problems which bears some resemblance to simulated annealing, a classical metaheuristic. Quantum annealers are special-purpose devices designed to run only this type of quantum algorithm. Other types of quantum algorithms are known that take on quite a different form, and are aimed at solving other types of problems. Quantum annealing can be applied to any optimization problem that can be expressed as a quadratic unconstrained binary optimization (QUBO) problem (Choi 2008;

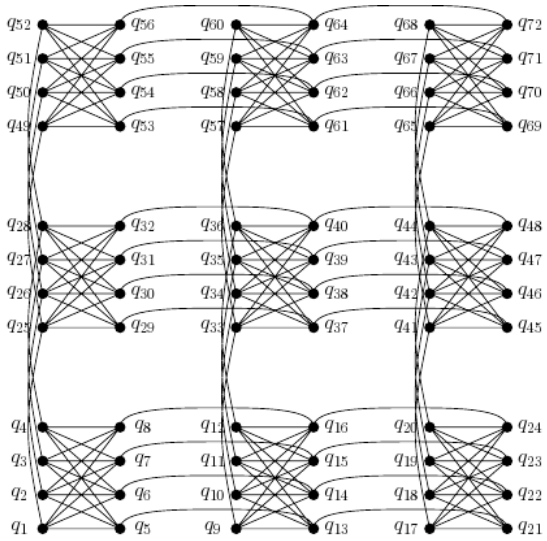

Figure 2: **The $(3,4)$-Chimera graph.** A schematic diagram from (Smelyanskiy et al. 2012) of the (M,L)-Chimera graph underlying D-Wave's architecture. In the $(3,4)$-Chimera graph shown, there are $M^2 = 9$ unit cells, each of which is a fully-connected bipartite graph $K_{4,4}$ containing $2L$ qubits. The qubits in the left column of each unit cell are connected to the analogous qubits in the unit cells above and below and the qubits in the right column of each unit cell are connected to the analogous qubits in the unit cells to the right and left.

Smelyanskiy et al. 2012; Lucas 2013). Quantum annealing is motivated by the possibility that quantum effects such as tunneling allow for efficient exploration of the cost-function landscape in ways unavailable to classical methods.

Input for quantum annealing: QUBO problems

QUBO problems are minimization problems with cost functions of the form

$$q(z_1,\ldots,z_N) = -\sum_{i=1}^{N} h_i z_i + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} J_{i,j} z_i z_j, \quad (1)$$

where the z_i are binary variables. A QUBO can be easily translated to an Ising Hamiltonian, the form of input a quantum annealer takes, through the linear transformation: $z_i = \frac{1}{2}(s_i + 1)$.

The simplicity of the QUBO formalism belies its expressivity. There exist many techniques for mapping more complicated problems to QUBO:

- Many optimization problems can be expressed in terms of cost functions that are polynomials over finite sets of binary variables. Any such function can be re-expressed, through degree-reduction techniques using ancilla variables, as quadratic functions over binary variables. We describe such degree-reduction technique in our section on the CNF mapping of planning problems to QUBO below.

- Cost functions involving non-binary, but finite-valued, variables can be rewritten in terms of binary variables alone, and optimization problems with constraints can often be written entirely in terms of cost functions over binary variables through the introduction of slack variables.

For these reasons, the QUBO setting is more general than it may, at first, seem.

Before describing the more complex QUBO mappings for general STRIPS planning problems, we give an example of a simple mapping from graph coloring to QUBO to give a feel of how mapping to QUBO works.

Example 1: Mapping of Graph Coloring, in which all vertices are to be colored so any two vertices connected by an edge have different colors, to QUBO.

Let $G = (V, E)$ be a graph with $n = |V|$ vertices, where E is the set of edges. The QUBO problem corresponding to the graph coloring problem with k colors on graph G, will have kn binary variables, z_{ic}, where $z_{ic} = 1$ means that vertex i is colored with color c, and $z_{ic} = 0$ means it is not.

The QUBO contains two different types of penalty terms. The first corresponds to the constraint that each vertex must be colored by exactly one color: $\sum_{c=1}^{k} z_{ic} = 1$. So for each vertex i, we have a term

$$\left(1 - \sum_{c=1}^{k} z_{ic}\right)^2.$$

The second corresponds to the constraint that two vertices connected by an edge cannot be colored with the same color. For each vertex i, we have a term

$$\sum_{(i,j)\in E} \sum_{c=1}^{k} z_{ic} z_{jc}.$$

Altogether, the QUBO is

$$\sum_{i=1}^{n} \left(1 - \sum_{c=1}^{k} z_{ic}\right)^2 + \sum_{(i,j)\in E} \sum_{c=1}^{k} z_{ic} z_{jc}.$$

Example 2: Mapping of Hamiltonian Path problem, in which the goal is a path that visits each vertex in a graph $G = (V, E)$ exactly once, to QUBO.

For a Hamiltonian path problem with n sites, we have n^2 variables

$$\{x_{11}, \ldots, x_{1n}, x_{21}, \ldots, x_{2n}, \ldots, x_{n1}, \ldots, x_{nn}.\}$$

The subscripted indices indicate, in order, a site and the time slot in which it is visited; $x_{ij} = 1$ means that the ith site is the jth site visited, and $x_{ij} = 0$ means that the ith site is not visited in the jth time slot.

There are three types of terms in the QUBO cost function. The first type of term enforces that each site is visited exactly once. Thus, for each site i, we will have a constraint:

$$\left(\sum_{j=1}^{n} x_{ij} - 1\right)^2.$$

The second type of term enforces that in each time slot no more than one site is visited. Thus, for each time slot j:

$$\left(\sum_{i=1}^{n} x_{ij} - 1\right)^2.$$

The third type of term is a single term penalizing the violation of edge constraints. It penalizes visiting the i'th site right after the ith site if they are not connected by an edge:

$$\sum_{j=1}^{j=n-1} \sum_{\{i,i'|(i,i')\notin E\}} x_{ij} x_{i',j+1}.$$

Embedding QUBOs in Specific Quantum Annealing Hardware

As outlined for planning in Figure 1, once an application instance has been mapped to a QUBO problem, a second step is required to compile it to the specific quantum annealing hardware that will be used. Typically, each quantum device has a set of physical quantum bits (qubits) that are linked in a certain way. Ideally, each binary variable z_i in a QUBO formula would be represented by a single qubit q_i of the machine. Hardware constraints place limits, however, on which qubits can be connected to which other qubits.

The strength of the coupling between two qubits q_i and q_j representing two binary variables z_i and z_j can model the term $J_{i,j} z_i z_j$ in the QUBO formula 1 introduced above. The D-Wave processors use a Chimera architecture in which each qubit is connected to at most 6 other qubits (Fig. 2), so any QUBO variable that appears in more than 6 terms must be represented by multiple physical qubits in order for the problem to be implemented in this architecture. The D-Wave Two used in the experiments has a $(8, 4)$-Chimera graph architecture, but with 3 broken qubits that are not used. Other limitations, beyond the degree 6 constraint, exist as well. Therefore, each logical qubit must be mapped to a connected set of physical qubits, which, together with the connecting edges, is called the logical qubit's *vertex-model*. The overall mapping of logical qubits and couplers to physical ones is called a *model*, as discussed below.

Consider, for example, a simple QUBO

$$z_1 z_2 + z_1 z_3 + z_2 z_3,$$

which can be represented as a triangle, with the three variables as the vertices, and the edge between each pair of vertices indicating a quadratic term of the QUBO. Ideally, we would represent each of these variables by qubits q_1, q_2, and q_3 with hardware connections between each pair so that the three terms in the QUBO can be directly realized in the hardware. Fig. 2 shows the qubit connections for the type of quantum annealing architecture we used in these experiments. In that graph, no three qubits are all mutually connected to each other. The best we can do is to use four qubits to represent the three variables z_1, z_2, and z_3. We may take, for example, z_1 to be represented by q_{52} and q_{56}, z_2 to be represented by q_{51}, and z_3 to be represented by q_{55}, so that there is a connection corresponding to each of the three

terms in the QUBO: the term $z_1 z_2$ can be implemented using the connection between qubits q_{56} and q_{51}, the term $z_1 z_3$ can be implemented using the connection between qubits q_{52} and q_{55}, and the term $z_1 z_2$ can be implemented using the connection between qubits q_{51} and q_{55}. We will also need to use the connection between qubits q_{52} and q_{56} to enforce that the two qubits take on the same value since together they are meant to represent a single variable.

Mapping General STRIPS Planning Problems to QUBO

In this section, we describe two different mappings from a general class of planning problems to QUBO. Specifically, we consider STRIPS planning problems, classical planning problems that are expressed in terms of binary state variables and actions. The first mapping takes a time-slice approach. The second approach first maps a planning problem to SAT, and then reduces higher order terms to quadratic terms through a series of gadgets. Our mappings allow both positive and negative preconditions.

Time-slice Mapping

This mapping from general classical planning problems to QUBO form is a variant of the one developed and described in (Smelyanskiy et al. 2012). This approach shares many similarities with existing compilation approaches (to SAT, CSP, MILP etc.) derived from the Plan Graph (Blum and Furst 1997). Specifically, it presets a horizon L and then encode the interleaving proposition and action layers up to the preset level L.

If the original planning problem has N state variables x_i and M actions y_j and we are looking for a plan of length L, then we define a time-slice QUBO problem in terms of $N(L+1)+LM$ binary variables. There are two groups of binary variables. The first group consists of $N(L+1)$ binary variables $x_i^{(t)}$ that indicate whether the state variable x_i is 0 or 1 at time step t, for $t \in \{0, \ldots, L\}$. The second group consists of LM binary variables $y_j^{(t)}$ that indicate whether or not the action y_j is carried out between time steps $t-1$ and t.

The total cost function is written as a sum

$$\begin{aligned} H &= H_{\text{initial}} + H_{\text{goal}} + H_{\text{precond}} \\ &\quad + H_{\text{effects}} + H_{\text{no-op}} + H_{\text{conflicts}}. \end{aligned}$$

The first two terms capture the initial condition and the goal condition. Let $\mathcal{I}^{(+)}$ be the set of state variables that are 1 in the initial condition and $\mathcal{I}^{(-)}$ be the set of state variables that are initially set to 0. Similarly, let $\mathcal{G}^{(+)}$ (resp. $\mathcal{G}^{(-)}$) be the set of goal variables with value 1 (resp. 0). To capture the requirement that a plan start in the appropriate initial state and meets the goals, we include in the cost function:

$$H_{\text{initial}} = \sum_{i \in \mathcal{I}^{(+)}} \left(1 - x_i^{(0)}\right) + \sum_{i \in \mathcal{I}^{(-)}} x_i^{(0)}$$

and

$$H_{\text{goal}} = \sum_{i \in \mathcal{G}^{(+)}} \left(1 - x_i^{(L)}\right) + \sum_{i \in \mathcal{G}^{(-)}} x_i^{(L)}.$$

We next add terms to the cost function that penalize a plan if an action is placed at time t but the prior state does not have the appropriate preconditions:

$$\begin{aligned} H_{\text{precond}} &= \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \mathcal{C}_j^{(+)}} \left(1 - x_i^{(t-1)}\right) y_j^{(t)} \right. \\ &\quad \left. + \sum_{i \in C_j^{(-)}} x_i^{(t-1)} y_j^{(t)} \right), \end{aligned}$$

where $\mathcal{C}_j^{(+)}$ is the set of positive preconditions for action j and $\mathcal{C}_j^{(-)}$ is the set of negative preconditions.

Next, we must penalize variable changes that are not the result of an action. We start with this term, the $H_{\text{no-op}}$ term, that penalizes variable changes:

$$H_{\text{no-op}} = \sum_{t=1}^{L} \sum_{i=1}^{N} \left[x_i^{(t-1)} + x_i^{(t)} - 2 x_i^{(t-1)} x_i^{(t)} \right].$$

This term gives a cost penalty of 1 every time a variable is flipped. Of course, when the effect of an action does result in a variable flipping, we do not want this penalty, so we will make up for this penalty when we add the term that corresponds to the effects of an action. Specifically, we need to penalize if the subsequent state does not reflect the effects of a given action. Let $\mathcal{E}_j^{(+)}$ be the set of positive effects for action j and $\mathcal{E}_j^{(-)}$ the set of negative effects. The penalty if the appropriate effects do not follow the actions is captured by the following term:

$$\begin{aligned} H_{\text{effects}} &= \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \mathcal{E}_j^{(+)}} y_j^{(t)} \left(1 + x_i^{(t-1)} - 2 x_i^{(t)}\right) \right. \\ &\quad \left. + \sum_{i \in \mathcal{E}_j^{(-)}} y_j^{(t)} \left(2 x_i^{(t)} - x_i^{(t-1)}\right) \right). \end{aligned}$$

In order to understand this term, we must consider it together with the no-op term. When $y_j^{(t)} = 1$, the corresponding term for $i \in \mathcal{E}_j^{(+)}$ (resp. $i \in \mathcal{E}_j^{(-)}$), taken together with the no-op term, can be written

$$\left(1 + 2 x_i^{(t-1)}\right) \left(1 - x_i^{(t)}\right)$$

(resp.

$$\left(3 - 2 x_i^{(t-1)}\right) x_i^{(t)}$$

for negative effects), resulting in a positive penalty unless $x_i^{(t)} = 1$ (resp. $x_i^{(t)} = 0$). By using this form we have corrected for the corresponding no-op term.

Parallel Plans: Classical planners often allow for parallel plans in which more than one action can take place at one

time if those actions could have been done in any order. Encodings that allow parallel plans are often significantly smaller due to the big reduction in the preset horizon value L. The QUBO encoding described so far works fine for domain with linear plans, but when more than one action can take place at a given time, we are in danger of overcorrecting for the no-op term. If multiple actions at the same time have the same effect, the H_{effects} term will add a term for each of those actions, thus overcompensating for the no-op penalty. To avoid overcompensating, we penalize multiple actions at the same time having the same effect, discouraging all such actions[1]. To ensure that two actions that conflict in the sense that positive preconditions of one overlap with negative effects of the other or vice versa, and to avoid overcompensating, we include the penalty

$$H_{\text{conflict}} = \sum_{t=1}^{L}\sum_{i=1}^{N}\left(\sum_{\left\{j\middle|i\in\mathcal{C}_j^{(+)}\cup\mathcal{E}_j^{(-)}\right\}}\sum_{\left\{j'\neq j\middle|i\in\mathcal{E}_{j'}^{(-)}\right\}} y_j^{(t)}y_{j'}^{(t)} + \sum_{\left\{j\middle|i\in\mathcal{C}_j^{(-)}\cup\mathcal{E}_j^{(+)}\right\}}\sum_{\left\{j'\neq j\middle|i\in\mathcal{E}_{j'}^{(+)}\right\}} y_j^{(t)}y_{j'}^{(t)}\right).$$

Encoding Size Improvements: While for explanatory purposes it was useful to include variables for the state at time $t = 0$, those specified by initial conditions can be set ahead of time, so that we don't need to include the H_{initial} term. The same is true of the H_{goal} term. We can also replace all of their occurrences in $H_{\text{no-op}}$, H_{precond}, and H_{effect} with these set values to simplify those constraints. Furthermore, reachability and relevant analysis starting from the initial and goal states, preprocessing techniques employed by compilation-based planners such as Blackbox (Kautz and Selman 1999) and GP-CSP (Do and Kambhampati 2001), can be used to remove or preset the values of variables in different layers. These simplifications result in modified terms $H'_{\text{no-op}}$, H'_{precond}, and H'_{effects}. Additionally, since in our setting we have followed the convention that preconditions must be positive, we can use a simpler version of the H_{precond} term:

$$H'_{\text{precond}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\sum_{i\in\mathcal{C}_j^{(+)}}\left(1-x_i^{(t-1)}\right)y_j^{(t)}.$$

For the scheduling problems we consider here, the QUBO simplifies to

$$H = H'_{\text{no-op}} + H'_{\text{precond}} + H'_{\text{effects}} + H_{\text{conflict}}.$$

CNF-based Mapping

Besides direct mapping, we also experimented with getting the QUBO encoding by first mapping planning problems to SAT (in CNF form) and then using the known approach to map the resulting CNF encoding to QUBO.

A SAT's conjunctive normal form (CNF) expression over n Boolean variables $\{x_i\}$ consists of a set of clauses $\{C_a\}$ each consisting of k variables, possibly negated, connected by logical ORs:

$$b_1 \vee b_2 \vee \cdots \vee b_k,$$

where

$$b_i \in \{x_1, x_2, ..., x_n, \neg x_1, \neg x_2, ..., \neg x_n\},$$

and the number of variables k in the clause can vary from clause to clause. In a CNF, all of the clauses must be satisfied, which means they are connected by an AND operator.

We used the first of the four PDDL to CNF translators built into the SATPLAN planner (Kautz 2004). This "action-based" encoding starts with the time-slice encoding approach and then further removes all variables representing state variables while adding constraints that capture the relationships between actions in consecutive time steps that were previously enforced by relationships between actions and state variables. We chose this encoding because it tends to produce the smallest SAT encodings. While it may not be the easiest to solve by a SAT solver, it's more likely to be translatable to a QUBO that can fit within our very limited number of available qubits in the D-Wave machine.

We convert a CNF instance to QUBO by first transforming it to Polynomial Unconstrained Binary Optimization (PUBO), a generalization of QUBO in which the objective function is a pseudo-Boolean of arbitrary degree. For each clause in a given CNF instance, we introduce a term to the PUBO instance equal to the conjunction of the negation of all the literals in that clause. Thus, an original negative literal is replaced by the corresponding binary variable and a positive literal is replaced by the difference of one and the corresponding binary variable. For example, the CNF clause $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$ would correspond to the PUBO term $(1 - x_1)x_2x_3(1 - x_4)$.

We then reduce higher degree terms in the PUBO instance using an iterative greedy algorithm that is related to one described in (Boros and Hammer 2002). At each step, the pair of variables that appears in the most terms is replaced by an ancilla variable corresponding to their conjunction. If there are multiple such pairs, then one is chosen arbitrarily. A penalty term is introduced to enforce that the ancilla variable indeed corresponds to the requisite conjunction. For example, to reduce the degree of a term $x_1x_2x_3$, we may introduce an ancilla variable y_{12} that we will encourage to equal x_1x_2 by using a penalty term $3y_{12}+x_1x_2-2x_1y_{12}-2x_2y_{12}$, which is 0 if $y_{12} = x_1x_2$ and > 0 otherwise. The term $x_1x_2x_3$ is removed from the PUBO and replaced with $y_{12}x_3 + 3y_{12} + x_1x_2 - 2x_1y_{12} - 2x_2y_{12}$. The penalty weight we use is equal to one plus the greater of the sums of the magnitudes of the positive coefficients and negative coefficients of the terms the ancilla is used to reduce (Babbush, O'Gorman, and Aspuru-Guzik 2013). The one is added to ensure that the constraint-satisfying solutions have lower total cost than the constraint-violating solutions. One is convenient, and in keeping with the integer coefficients

[1] A less stringent way to avoid overcompensating would be to add this penalty only when the effect changes the variable, as we have done in the no-op term. The problem is that natively that is not a quadratic term. Of course one could then reduce that term, but here we choose to use the more stringent solution.

for the other terms, but any positive constant would do. This procedure is repeated until the resulting PUBO is quadratic.

Experimental Setup

To evaluate our approach, we use the benchmark set of planning instances based on graph-coloring (Rieffel et al. 2014a) that consist of parametrized families of hard planning problems. Having parametrized families enables the investigation of scaling behavior using small problems, which is crucial for evaluating early technology that is not mature enough to run real-world problems. Even when setting the encoding horizon of the graph-coloring to 1 to fit them onto the 512 qubits available in current quantum annealers, the problems can still be considered hard. Note that vertex coloring, an NP-complete problem, is strongly related to the core scheduling aspect of many planning applications (Chien et al. 2012). A scheduling problem in which no pair of tasks can be assigned the same time-slot is analogous to a coloring instance in which the the tasks are vertices, conflicts are edges, and the minimum makespan is the chromatic number, i.e. the minimum number of colors necessary to color each of the vertices such that no two adjacent ones have the same color.

Because of the overhead in mapping and embedding planning problems, even the smallest IPC problems (IPC 2004) are more than an order of magnitude too large to be run on the current D-Wave device. Therefore, we currently do not include the any result on existing IPC benchmarks.

Vertex Coloring as Planning

Given an undirected graph $G = \{V, E\}$ with n vertices and k colors, the vertex coloring problem asks for a solution in which: (1) all vertices are colored and (2) any pair of vertices connected by an edge is colored differently. The corresponding planning problem is as follows: for each vertex v there are:

- k actions a_v^c representing coloring v with color c;
- A 'goal variable' s_v^g representing whether or not v has been colored at all; and
- A state variable s_v^c representing whether or not v has been colored with the color c.

Let $C(v)$ be the set of neighboring vertices that are connected to v by an edge. For each action a_v^c, there are $|C(v)|+1$ preconditions: (1) $s_v^g = F$, which indicates that v is not already colored; and (2) for each $w \in C(v)$, $s_w^c = F$, guaranteeing that none of neighboring v_i are already colored with color c.

Each action a_v^c has two effects: $s_v^g = T$ and $s_v^c = T$.

In the initial state, none of the vertices are colored: $\forall v \in V, \forall c \in [k] : s_v^g = F$, and $s_v^c = F$. The goal state requires that all vertices are colored: $\forall v \in V : s_v^g = T$. A plan is a sequence of n actions, each of which colors a vertex v.

Problem generation: We parametrically generate instances using Erdös-Rényi model of random graphs $G_{n,p}$, where n is the number of vertices and p is the probability of an edge between each pair of vertices, using an extension of Culberson et al.'s (Culberson, Beacham, and Papp 1995) graph generator program. Our extension generates PDDL files (Rieffel et al. 2014b), at the phase transition $c = pn = 4.5$ (Achlioptas and Friedgut 1999; Dubois and Mandler 2002; Achlioptas and Moore 2003; Coja-Oghlan 2013).

We preset the number of colors to $k = 3$ and for that k value the maximum problem size that we can embed in the D-Wave Two machine with 509 qubits is $n = 16$. Specifically, for each of $n = 8, 9, \ldots, 16$, we use 100 solvable problem instances at the phase transition for each size. For $n = 12$ to $n = 16$, we reuse problems generated in (Rieffel et al. 2014a), and for $n = 8$ to $n = 11$, we generate new ones using the same approach.

For each generated instance, we then generate 3 different QUBOs, each described in the previous sections: (i) direct mapping; (ii) time-slice mapping; and (iii) CNF-based mapping.

Embedding: From a mapped QUBO instance, we generate a vertex model by running D-Wave's heuristic embedding software (Cai, Macready, and Roy 2014) on the mapped QUBO instance, using the software's default parameters. The output of the embedding software is a set of pairwise-disjoint, connected vertex models $\{C_i\}$ in the hardware graph corresponding to the variables $\{z_i\}$ in the original QUBO, which will then be converted to Ising form to be run on the D-Wave machine. Before running, the Ising is rescaled so that all coefficients are between $[-1, 1]$. We performed our own parameter setting following (Rieffel et al. 2014b), rather than using D-Wave's defaults. The parameter settings for these runs are discussed in detail in (Rieffel et al. 2014b).

Solving: All quantum annealing runs were performed on the 509-qubit D-Wave Two machine housed at NASA Ames. While debate continues as to the best physical model for D-Wave machines (Johnson et al. 2011b; Boixo et al. 2013; Smolin and Smith 2013; Wang et al. 2013; Boixo et al. 2014; Shin et al. 2014b; Vinci et al. 2014; Shin et al. 2014a), these machines provide the first opportunity for researchers to experiment with quantum annealing. In all cases, we used an annealing time of 20 μsec, the shortest available on the hardware, though evidence suggests that a shorter time may be optimal for problems of the present size. For each embedded QUBO instance, we performed $45,000$ anneals using each of ten gauges (i.e. local symmetry transformations that leave the objective function invariant but physically change the effect of biases (Perdomo-Ortiz et al. 2015)), for a total of $450,000$ anneals per QUBO instance.

Because all of the problems we consider are solvable, we know the ground state energy (i.e., optimal value for the function q in Equation 1) in all cases; zero, the minimal value of the QUBO in all cases is attainable, and from that we can compute the ground state energy of the embedded Ising problem that was actually run. (Even for unsolvable instances, the quantum annealer would return solutions in exactly the same way it does when it fails to find a schedule when it exists.) For each embedded instance, once we obtain

the $450,000$ results from the run, we check how many times the ground state energy was obtained, which gives us the probability of solution r for a 20 μsec anneal. We then compute the expected number of runs $R = \frac{\ln(1-0.99)}{\ln(1-r)}$ required to obtain a 99% success probability, multiply by the anneal time of 20 μsec, and report $20 \times R$ μsec, the expected total anneal time to obtain a 99% success probability. We are effectively using a 0.9 sec. cutoff time, since the expected anneal time when only one anneal solves is 0.9 secs. Given that classical planners solve these problems in less than 0.1 secs., with the best planners for these problems solving them in less than 0.01 secs. (Rieffel et al. 2014a), this cutoff time seems reasonable.

We report the median expected total anneal time across 100 instances, with error bars corresponding to the 35th and 65th percentiles. Thus each data point shown represents 45 million anneals. While the total annealing time for each point is only 90 seconds, because the process to read-out the state of all qubits (i.e. solution extraction) takes considerably longer than the anneal time, and because of shared use of the machine, the wall clock time to obtain a single data point is hours not minutes. Finding the embedding, by far the longest step in the process, can take minutes for the largest instances, but fortunately needs to be performed only once per QUBO instance.

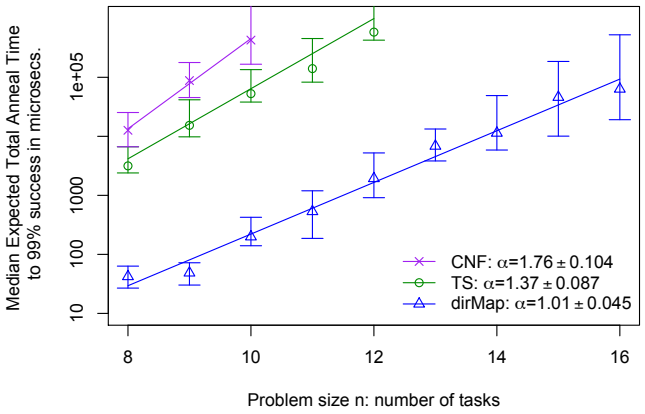

Figure 3: **Comparison of the median expected total anneal time to 99% percent success for the two mappings.** Each data point shows the median expected total annealing time to achieve 99% success over the 100 problems of each size given on the x-axis. The error bars are at the 35th and 65th percentiles. When at least half of the instances do not solve within the 0.9 sec. effective cutoff time, we no longer show the point. Also, when fewer than 65% solve, the top of the error bar is indeterminate, as happened for the last point shown in both the CNF and time-slice series.

Results and Analysis

Fig. 3 shows the relative performance, in terms of median expected total annealing time for 99% percent success, of the D-Wave Two on the family of graph coloring-based planning problems described above. When at least half of the instances do not solve within the 0.9 sec. effective cutoff time, we no longer show the point. For the CNF mapping, that happens by problem size 11. For the time-slice instances, at least half do not solve within the cutoff time by problem size 13. The figure also shows the performance using a direct map of graph coloring to QUBO.This direct mapping performs better than both of the general mappings for planning problems; it is more compact (due to its being specific to this type of problem) and likely benefits from an homogeneous parameter setting (Venturelli et al. 2014), as it generates a more uniform distribution of vertex model sizes (see Fig. 5) than the other two mappings.

There is a substantial difference between the performance on the time-slice instances and the CNF instances, with the median expected total annealing time to achieve 99% success being about a factor of 5 greater for the CNF instances than the time-slice instances (Fig. 3). The scaling for the time-slice approach is also significantly better than for the CNF approach, with an α value of 1.37 rather than 1.76 (though the scaling is estimated on very few data points).

QUBO size: (Rieffel et al. 2014b) compared some straightforward properties of both the mapped and embedded QUBOs for the two mappings, but these simple properties were all sufficiently similar across the two mappings that they could not account for so marked a difference in performance. In summary, the time-slice and CNF mappings yield comparably-sized QUBOs, with similar numbers of couplings. For problem size n, the time-slice mapping yields a QUBO of size $8n$ qubits. The CNF approach yields variable size mapped QUBOs, with the median size CNF QUBO over 100 problems only 4–8 qubits larger than the median size of the time-slice QUBOs for problem sizes 8–12. This slight difference in size cannot account for the difference in performance because for larger size problems, when the time-slice QUBOs begin to exceed the CNF QUBOs in size, the performance of the former is still better. Similarly, the median number of couplings for the CNF QUBOs exceeds that of the time-slice QUBOs by only 8–16 for problem sizes 8–12.

The most obvious properties of the embedded QUBOs are also similar. The median embedding sizes of the CNF QUBOs are only 7–28 qubits larger than the embedded time-slice QUBOs in this range, no more than a 10% difference. Large embedded vertex models contribute to poor performance, but the median (over the 100 problems) average vertex model size, and the median 90th percentile vertex model size of the embedded QUBOs for the two different mappings are virtually indistinguishable. A small difference between the median maximum vertex model sizes is seen, but it is not statistically significant. Furthermore, throughout the size range tested, the median median vertex model size – the median over the 100 problem instances of the median vertex model size of each instance – and even the median 65th percentile vertex model size, for both mappings is 1.

Deeper analysis: We took a deeper look at the distributions

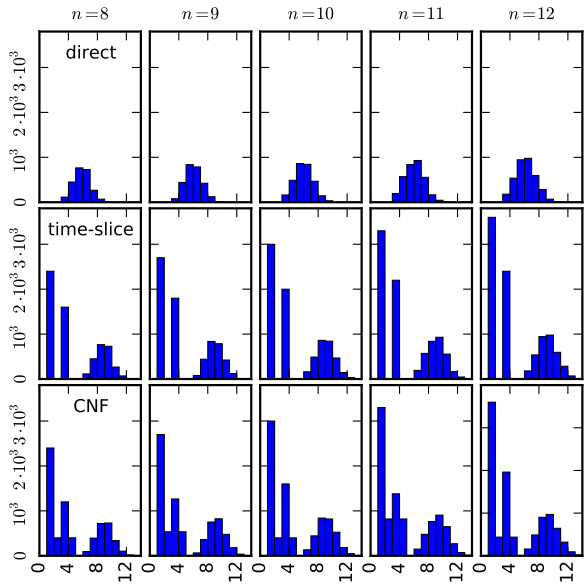

Figure 4: **Vertex degree histogram.** Histograms of the vertex degrees for the mapped QUBO graphs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

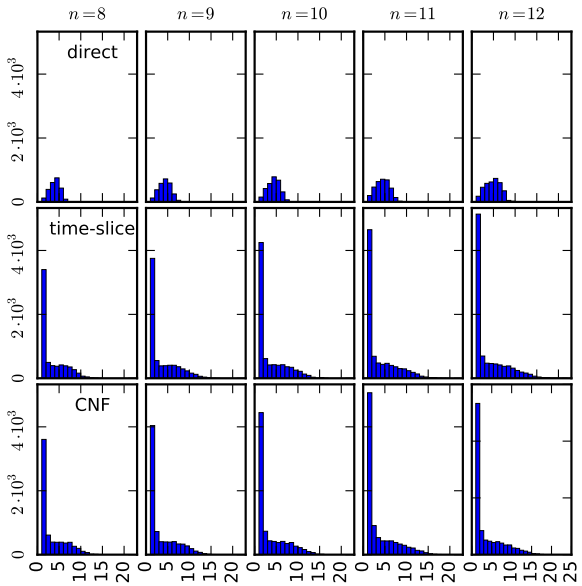

Figure 5: **Vertex model size histogram.** Histograms of the vertex model sizes in the embedded QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

of various simple properties related to the mapped and embedding QUBOs arising from the two mappings. We also show the distributions for the direct map for comparison. In the mapped QUBOs, we looked at the distribution of of vertex degrees (the number of quadratic terms in which a variable z_i appears in the mapped QUBO). As can be seen in Fig. 4, the histograms for the time-slice and CNF mappings are very similar, while they differ markedly from the histograms for the direct map. In the embedded QUBOs, we looked at the distribution of the vertex model sizes (Fig. 5) and also the distribution of the graph diameter of the vertex models (not shown), but found little difference between the distributions for the time-slice and CNF embedded QUBOs. Therefore these properties can contribute at most a small amount to the performance difference between the two mappings.

We begin to see differences when we look at distributions of the coefficients in the mapped QUBOs. Fig. 6 shows histograms of the coefficient of the mapped QUBOs (Equation 1) , converted to Ising, and rescaled so that all of the h_i and J_{ij} coefficients are between $[-1, 1]$. Let $j_i = \sum_j J_{ij}$. Fig. 7 shows a histogram of the h_i and j_i. Both histograms show significant differences between the two mappings.

Another potential origin of the performance difference is the topology of the vertex models. A quick analysis showed that for all three mappings nearly all ($> 99\%$) of the vertex models of the embedded QUBOs are trees. We intend to do a further classification of the graph structures, and to examine differences in the frequency of different structures between the mappings.

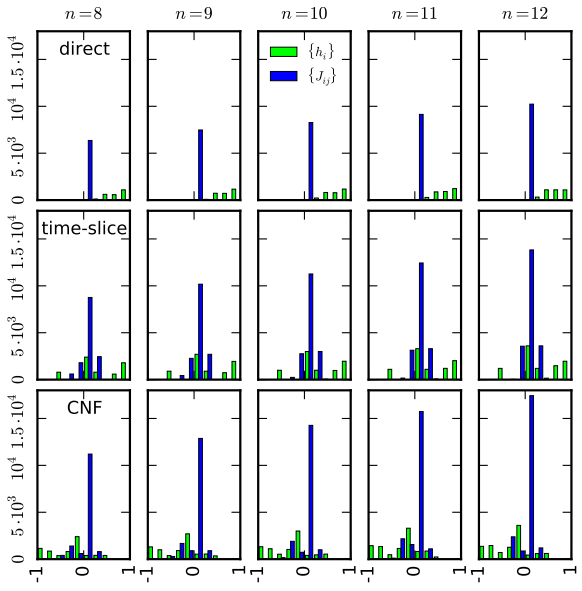

Figure 6: **QUBO coefficient histogram.** Histograms of the h_i and J_{ij} in the mapped QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

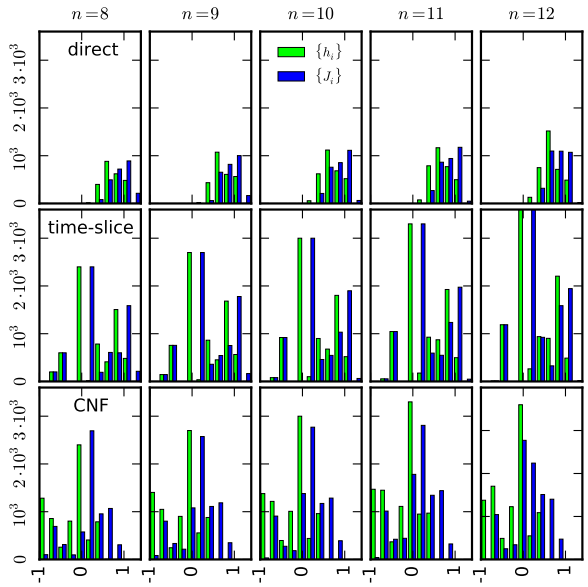

Figure 7: **Histogram of h_i and $j_i = \sum_j J_{ij}$.** Histograms of the h_i and $j_i = \sum_j J_{ij}$ in the mapped QUBOs for each problem size under the three mappings: direct mapping, time-slice mapping, and CNF mapping.

Conclusions and Future Work

We show how quantum annealing can solve planning problems via mapping to QUBO. We introduced two general mapping techniques and applied them to planning problems based on graph-coloring. We ran these problems on an early quantum annealer and saw significant performance differences. We began an investigation of various properties of the mapped and embedded QUBOs to understand which properties do and do not contribute to the performance differences.

In the future, we will examine the differences in distribution of coefficients and the topology of vertex-models to generate hypotheses regarding properties that could explain the performance differences. To test these hypotheses, we will generate small, artificial instances capturing those properties and evaluate the annealer's performance. For the properties that pass this initial test, we will perform a statistical analysis of the correlation between them and the performance of the annealer on the family of scheduling-type problems.

While at this early stage there is no advantage in solving STRIPS planning problems via quantum annealing over classical compilation approaches such as SAT, CSP, or MILP, we believe quantum annealing, and especially compilation techniques for quantum annealing, are both worth exploring even on primitive quantum hardware for several reasons. First, certain quantum algorithms have been proven to outperform classical algorithms on classes of problems of practical interest, sometimes, as for factoring, reducing the complexity from superpolynomial to polynomial. Many of the most useful classical algorithms in use today are heuristic algorithms, which have not been mathematically proven to outperform other approaches, but have been shown to be more effective empirically. Until recently, it was not possible to explore existing quantum heuristic algorithms, because without quantum hardware an empirical analysis could not be done. One of the biggest open questions in quantum computing is the breadth of its applications, with the potential of heuristic quantum algorithms, such as quantum annealing, being the biggest unknown. Major hardware development efforts are underway to build better quantum computational hardware. In order to fully explore the potential of this hardware, we must understand how best to compile practical problems to a form that is suitable for quantum hardware.

While early quantum annealing hardware can handle only small instances, by analyzing the results obtained under these limitations, we can nevertheless gain insights into the best programming and compilation techniques for quantum annealers, and ultimately into the potential of quantum annealing to solve problems of practical interest in planning and scheduling and beyond.

Acknowledgements

The authors are grateful to Zhihui Wang for helpful discussions and feedback on the draft, and to Vadim Smelyanskiy for useful discussions and support. This work was supported in part by the Office of the Director of National Intelligence (ODNI), the Intelligence Advanced Research Projects Activity (IARPA), via IAA 145483; by the AFRL Information Directorate under grant F4HBKC4162G001. 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 ODNI, IARPA, AFRL, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon. The authors also would like to acknowledge support from the NASA Advanced Exploration Systems program and NASA Ames Research Center and NASA grant NNX12AK33A.

References

Achlioptas, D., and Friedgut, E. 1999. A sharp threshold for k-colorability. *Random Structures and Algorithms* 14(1):63–70.

Achlioptas, D., and Moore, C. 2003. Almost all graphs with average degree 4 are 3-colorable. *Journal of Computer and System Sciences* 67(2):441–471.

Babbush, R.; O'Gorman, B.; and Aspuru-Guzik, A. 2013. Resource efficient gadgets for compiling adiabatic quantum optimization problems. *Annalen der Physik* 525(10-11):877–888.

Blum, A., and Furst, M. 1997. Planning through planning graph analysis. *Artificial Intelligence Journal* 90:281–330.

Boixo, S.; Albash, T.; Spedalieri, F. M.; Chancellor, N.; and Lidar, D. A. 2013. Experimental signature of programmable quantum annealing. *Nature communications* 4.

Boixo, S.; Rønnow, T. F.; Isakov, S. V.; Wang, Z.; Wecker, D.; Lidar, D. A.; Martinis, J. M.; and Troyer, M. 2014. Evidence for quantum annealing with more than one hundred qubits. *Nature Physics* 10(3):218–224.

Boros, E., and Hammer, P. L. 2002. Pseudo-boolean optimization. *Discrete applied mathematics* 123(1):155–225.

Cai, J.; Macready, B.; and Roy, A. 2014. A practical heuristic for finding graph minors. arXiv:1406:2741.

Chien, S.; Johnston, M.; Frank, J.; Giuliano, M.; Kavelaars, A.; Lenzen, C.; Policella, N.; and Verfailie, G. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *12th International Conference on Space Operations*.

Choi, V. 2008. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing* 7(5):193–209.

Coja-Oghlan, A. 2013. Upper-bounding the k-colorability threshold by counting covers. arXiv:1305.0177.

Culberson, J.; Beacham, A.; and Papp, D. 1995. Hiding our colors. In *Proceedings of the CP95 Workshop on Studying and Solving Really Hard Problems*, 31–42.

Das, A., and Chakrabarti, B. K. 2008. Colloquium: Quantum annealing and analog quantum computation. *Rev. Mod. Phys.* 80:1061–1081.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence Journal* 132(2):151–182.

Dubois, O., and Mandler, J. 2002. On the non-3-colourability of random graphs. arXiv:math/0209087.

Farhi, E.; Goldstone, J.; Gutmann, S.; and Sipser, M. 2000. Quantum computation by adiabatic evolution. arXiv:quant-ph/0001106.

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

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

2004. The international planning competition website. http://icaps-conference.org/index.php/Main/Competitions.

Johnson, M. W.; Amin, M. H. S.; Gildert, S.; and et al. 2011a. Quantum annealing with manufactured spins. *Nature* 473:194–198.

Johnson, M.; Amin, M.; Gildert, S.; Lanting, T.; Hamze, F.; Dickson, N.; Harris, R.; Berkley, A.; Johansson, J.; Bunyk, P.; et al. 2011b. Quantum annealing with manufactured spins. *Nature* 473(7346):194–198.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *Proceedings of IJCAI'1999*.

Kautz, H. 2004. Satplan04: Planning as satisfiability. *Working Notes on the Fourth International Planning Competition (IPC-2004)* 44–45.

Lucas, A. 2013. Ising formulations of many NP problems. arXiv:1302.5843.

Nielsen, M., and Chuang, I. L. 2001. *Quantum Computing and Quantum Information*. Cambridge: Cambridge University Press.

Perdomo-Ortiz, A.; Fluegemann, J.; Biswas, R.; and Smelyanskiy, V. N. 2015. A performance estimator for quantum annealers: Gauge selection and parameter setting. *arXiv preprint arXiv:1503.01083*.

Rieffel, E. G., and Polak, W. 2011. *A Gentle Introduction to Quantum Computing*. Cambridge, MA: MIT Press.

Rieffel, E. G.; Venturelli, D.; Hen, I.; Do, M.; and Frank, J. 2014a. Parametrized families of hard planning problems from phase transitions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14)*, 2337 – 2343.

Rieffel, E. G.; Venturelli, D.; O'Gorman, B.; Do, M.; Prystay, E.; and Smelyanskiy, V. N. 2014b. A case study in programming a quantum annealer for hard operational planning problems. To Appear in Quantum Information Processing, and arXiv:1407.2887.

Shin, S. W.; Smith, G.; Smolin, J. A.; and Vazirani, U. 2014a. Comment on "Distinguishing classical and quantum models for the D-Wave device". arXiv:1404.6499.

Shin, S. W.; Smith, G.; Smolin, J. A.; and Vazirani, U. 2014b. How "quantum" is the D-Wave machine? arXiv:1401.7087.

Smelyanskiy, V. N.; Rieffel, E. G.; Knysh, S. I.; Williams, C. P.; Johnson, M. W.; Thom, M. C.; Macready, W. G.; and Pudenz, K. L. 2012. A near-term quantum computing approach for hard computational problems in space exploration. arXiv:1204.2821.

Smolin, J. A., and Smith, G. 2013. Classical signature of quantum annealing. arXiv:1305.4904.

Venturelli, D.; Mandrà, S.; Knysh, S.; O'Gorman, B.; Biswas, R.; and Smelyanskiy, V. 2014. Quantum optimization of fully-connected spin glasses. *arXiv preprint arXiv:1406.7553*.

Vinci, W.; Albash, T.; Mishra, A.; Warburton, P. A.; and Lidar, D. A. 2014. Distinguishing classical and quantum models for the D-Wave device. arXiv:1403.4228.

Wang, L.; Rønnow, T. F.; Boixo, S.; Isakov, S. V.; Wang, Z.; Wecker, D.; Lidar, D. A.; Martinis, J. M.; and Troyer, M. 2013. Comment on "Classical signature of quantum annealing".

Logic-based Benders Decomposition for Planning and Scheduling: A Computational Analysis*

André A. Ciré
University of Toronto, Canada
AndreCire@rotman.utoronto.ca

Elvin Çoban
Özyeğin University, Istanbul, Turkey
elvin.coban@ozyegin.edu.tr

J. N. Hooker
Carnegie Mellon University, Pittsburgh, USA
jh38@andrew.cmu.edu

Abstract

Logic-based Benders decomposition (LBBD) has improved the state of the art for solving a variety of planning and scheduling problems, in part by combining the complementary strengths of constraint programming (CP) and mixed integer programming (MIP). We undertake a computational analysis of specific factors that contribute to the success of LBBD, to provide guidance for future implementations. We study a problem class that assign tasks to multiple resources and poses a cumulative scheduling problem on each resource. We find that LBBD is at least 1000 times faster than state-of-the-art MIP on larger instances, despite recent advances in the latter. Further, we conclude that LBBD is most effective when the planning and scheduling aspects of the problem are roughly balanced in difficulty. The most effective device for improving LBBD is the inclusion of a subproblem relaxation in the master problem. The strengthening of Benders cuts also plays an important role when the master and subproblem complexity are properly balanced. These findings suggest future research directions.

Introduction

Logic-based Benders decomposition (LBBD) is a generalization of classical Benders decomposition that can be applied to a much wider variety of combinatorial optimization problems. LBBD is particularly attractive for planning and scheduling, where it can combine mixed integer programming (MIP) and constraint programming (CP) in a way that exploits their relative strengths. Implementations of this method have obtained computational results superior to those of state-of-the-art MIP and CP solvers, sometimes by several orders of magnitude

However, a computational analysis of the precise factors responsible for this success has never been conducted. In particular, applications of LBBD typically strengthen the Benders cuts in various ways, as well as including a subproblem relaxation in the master problem, on the assumption that these techniques improve performance. We undertake here a systematic study of their actual impact. In addition, because MIP technology has improved markedly in recent years, we compare LBBD with a recent state-of-the-art commercial MIP solver (CPLEX) to determine whether the advantage of LBBD persists.

We focus attention on a basic planning and scheduling problem in which tasks are assigned to resources and then scheduled on those resources. The tasks assigned to a given resource can run concurrently so long as the total rate of resource consumption never exceeds a given maximum (cumulative scheduling). The assignment problem is solved by MIP and the scheduling problem by CP.

Previous Work

Logic-based Benders decomposition was introduced by Hooker (1995) and Hooker and Yan (1995), and a general theory was presented in Hooker (2000) and Hooker and Ottosson (2003). A guide to applying LBBD to planning and scheduling problems is provided in Hooker (2007b) and Hooker (2012).

Classical Benders decomposition derives Benders cuts from dual or Lagrange multipliers in the subproblem (Benders 1962; Geoffrion 1972). However, this presupposes that the subproblem is a linear or nonlinear programming problem. Logic-based Benders decomposition has the advantage that Benders cuts can, at least in principle, be obtained from a subproblem of any form by solving its *inference dual* (Hooker 1996). The solution of the dual is a *proof* of optimality for fixed values of the master problem variables (whence the name "logic-based"). The core idea of Benders decomposition is that *this same proof* may establish a bound on the optimal value when the master problem variables take other values. The corresponding Benders cut enforces this bound in the master problem.

Logic-based Benders cuts must be designed specifically for each class of problems, but this provides an opportunity to exploit problem structure. The Benders framework is also natural for combining MILP and CP, because one method can be used to solve the master problem and the other the subproblem. This is particularly advantageous when the subproblem is a scheduling problem, for which CP methods are well suited (Baptiste, Pape, and Nuijten 2001; Hooker 2007a). The combinatorial nature of the scheduling problem is no longer a barrier to generating Benders cuts.

The computational advantages of LBBD have been demonstrated in a number of studies, including Benini et al. (2005), Cambazard et al. (2004), Chu and Xia (2004),

*This work was partially supported by NSF Grant 1130012 and AFOSR grant FA9550-11-1-0180.

Çoban and Hooker (2013), Corréa, Langevin, and Rousseau (2004), Fazel-Zarandi and Beck (2009), Harjunkoski and Grossmann (2001; 2002), Hooker (2004; 2005a; 2005b; 2006; 2007b), Jain and Grossmann (2001), Maravelias and Grossmann (2004), Terekhov, Beck, and Brown (2007), Thorsteinsson (2001), Timpe (2002), and Yunes, Aron, and Hooker (2010).

We presented some of the computational results given here at a recent CPAIOR conference (Ciré, Çoban, and Hooker 2013). In this paper we present more detailed results, including performance profiles, iteration counts, a computation time breakdown by master and subproblem, and the peformance of Benders cuts without a subproblem relaxation. We also analyze the implications of these results.

Fundamentals

Logic-based Benders decomposition is based on the concept of an *inference dual*. Consider an optimization problem

$$\begin{array}{l} \min\ f(x) \\ C(x) \\ x \in D \end{array} \tag{1}$$

where $C(x)$ represents a constraint set containing variables x, and D is the domain of x (such as $\mathbb{R}^n$ or $\mathbb{Z}^n$). The inference dual is the problem of finding the tightest lower bound on the objective function that can be deduced from the constraints:

$$\begin{array}{l} \max\ v \\ C(x) \overset{P}{\vdash} (f(x) \geq v) \\ v \in \mathbb{R},\ P \in \mathcal{P} \end{array} \tag{2}$$

Here $C(x) \overset{P}{\vdash} (f(x) \geq v)$ indicates that proof P deduces $f(x) \geq v$ from $C(x)$. The domain of variable P is a family $\mathcal{P}$ of proofs, and the dual solution is a pair (v, P). When the primal problem (1) is a feasibility problem with no objective function, the dual can be viewed as the problem finding a proof P of infeasibility.

If problem (1) is a linear programming (LP) problem $\min\{cx \mid Ax \geq b,\ x \geq 0\}$, the inference dual becomes the classical LP dual (assuming feasibility) for an appropriate proof family $\mathcal{P}$. Namely, each proof P corresponds to a tuple $u \geq 0$ of multipliers, and P deduces the bound $cx \geq v$ when the surrogate $uAx \geq ub$ dominates $cx \geq v$; that is, $uA \leq c$ and $ub \geq v$. The dual therefore maximizes v subject to $uA \leq c$, $ub \geq v$, and $u \geq 0$. Equivalently, it maximizes ub subject to $uA \leq c$ and $u \geq 0$, which is the classical LP dual.

Logic-based Benders decomposition applies to problems of the form

$$\begin{array}{l} \min\ f(x,y) \\ C(x,y) \\ x \in D_x,\ y \in D_y \end{array} \tag{3}$$

Fixing x to $\bar{x}$ defines the subproblem

$$\begin{array}{l} \min\ f(\bar{x},y) \\ C(\bar{x},y) \\ y \in D_y \end{array} \tag{4}$$

Let proof P solve the inference dual of the subproblem by deducing the bound $f(\bar{x},y) \geq v^*$. A Benders cut $v \geq B_{\bar{x}}(x)$ is derived by identifying a bound $B_{\bar{x}}(x)$ that the same proof P deduces for any given x. Thus, in particular, $B_{\bar{x}}(\bar{x}) = v^*$. The kth master problem is

$$\begin{array}{l} \min\ v \\ v \geq B_{x^i}(x),\ i = 1,\dots,k-1 \\ x \in D_x \end{array} \tag{5}$$

where $x^1,\dots,x^{k-1}$ are the solutions of the first $k-1$ master problems. If the subproblem is a feasibility problem with no objective function, the Benders cut is a constraint that excludes $\bar{x}$ and perhaps other solutions that proof P shows to be infeasible.

At this point we solve the master problem and set $\bar{x}$ equal to an optimal solution of the master, whereupon the process repeats. The algorithm terminates when the optimal value v_k of the master problem is equal to the minimum of $B_{x^i}(x^i)$ over $i = 1,\dots,k$. At any stage of the algorithm, v_k is a lower bound on the optimal value of (3), and each $B_{x^i}(x^i)$ is an upper bound.

Classical Benders decomposition is the result of applying logic-based Benders decomposition to a problem of the form

$$\begin{array}{l} \min\ f(x) + cy \\ g(x) + Ay \geq b \\ x \in D_x,\ y \geq 0 \end{array} \tag{6}$$

The subproblem is an LP:

$$\begin{array}{l} \min\ f(\bar{x}) + cy \\ Ay \geq b - g(\bar{x}) \\ y \geq 0 \end{array} \tag{7}$$

whose inference dual is the LP dual. Its solution u defines a surrogate $uAy \geq u(b - g(\bar{x}))$ that dominates $cy \geq v^*$ and therefore deduces that $f(\bar{x}) + cy \geq f(\bar{x}) + u(b - g(\bar{x}))$. The same u deduces $f(x) + cy \geq f(x) + u(b - g(x))$ for any x, and we have the classical Benders cut $v \geq f(x) + u(b - g(x))$. When the subproblem is infeasible, the dual has an extreme ray solution u that proves infeasibility because $uA \leq 0$ and $u(b - g(x)) > 0$. The Benders cut is therefore $u(b - g(x)) \leq 0$.

In practice, the solution of the subproblem inference dual is the proof of optimality obtained while solving the subproblem. The simplest type of Benders cut is a *nogood cut*, which states that the solution of the subproblem cannot be improved unless at least one x_j is fixed to a different value. This yields a nogood cut

$$v \geq \begin{cases} v^* & \text{if } x_j = \bar{x}_j \text{ for all } j \in J \\ -\infty & \text{otherwise} \end{cases} \tag{8}$$

where J is the index set for the x_js. If the subproblem is infeasible, the nogood cut states simply that $x_j \neq \bar{x}_j$ for some $j \in J$.

Nogood cuts can be strengthened in various ways. The simplest examines the dual proof and observes that only the variable settings $x_j = \bar{x}_j$ for $j \in \bar{J}$ appear as premises. This yields a strengthened nogood cut by replacing J with $\bar{J}$ in

(8). If the dual proof is not accessible, we can strengthen the cuts by re-solving the subproblem when some of the premises $x_j = \bar{x}_j$ are dropped, and checking whether the optimal subproblem value is the same. Further analysis of the optimality proof may yield *analytic* Benders cuts that provide useful bounds when $x_j \neq \bar{x}_j$ for some of the $j \in J$.

LBBD for Planning and Scheduling

The problem is to assign each of n tasks to one of m resources so as to minimize cost, subject to the condition that the tasks assigned to each resource can be feasibly scheduled on that resource. Each task j must be processed within time window $[L_j, U_j]$, has processing time p_{ij} on resource i, and consumes resource i at the rate r_{ij}. The total rate of resource i consumption can be at most L_i. It is assumed that there is a fixed cost c_{ij} for processing task j on resource i.

The master problem variables x become binary variables δ_{ij}, where δ_{ij} is 1 when task j is assigned to resource i. The master problem (5) is

$$\begin{aligned} &\min \sum_{ij} c_{ij}\delta_{ij} \\ &\sum_i \delta_{ij} = 1, \quad j = 1, \ldots, n \\ &\text{Benders cuts} \\ &\text{Subproblem relaxation} \\ &\delta_{ij} \in \{0,1\}, \quad \text{all } i, j \end{aligned} \tag{9}$$

The subproblem decouples into a separate scheduling problem for each resource. The subproblem variables y are the start time s_j of task j. Let J_i be the set of tasks assigned to resource i in the solution of the master problem. Let the constraint cumulative(s, p, r, R) require that tasks be scheduled at start times $s = (s_1, \ldots, s_k)$ so that the resource consumption rate never exceeds R. The subproblem for resource i is the feasibility problem

$$\begin{aligned} &L_j \leq s_j \leq U_j - p_{ij}, \quad \text{all } j \in J_i \\ &\text{cumulative}((s_j, j \in J_i), (p_{ij}, j \in J_i), (r_{ij}, j \in J_i), R_i) \end{aligned} \tag{10}$$

We generate a Benders cut for each resource for which the assignment is infeasible. A simple nogood cut is as described above, which in the present context becomes

$$\sum_{j \in J_i} (1 - \delta_{ij}) \geq 1 \tag{11}$$

Stronger cuts would ideally be obtained by examining the proof of infeasibility when the subproblem is solved. For example, if infeasibility is proved by edge finding (Baptiste, Pape, and Nuijten 2001), or edge finding plus branching, one could observe the set $\bar{J}_i$ of tasks that actually play a role in the proof, and replace J_i with $\bar{J}_i$ in (11). Unfortunately, this kind of information is not available from the commercial CP software used to solve the scheduling problem. We therefore strengthen the nogood cut heuristically as indicated earlier. We remove elements from J_i one at a time and re-solve the subproblem until the scheduling problem becomes feasible. We then restore the last element removed, and the resulting $\bar{J}_i$ becomes the basis for the strengthened nogood cut (Hooker, 2005a; 2007c).

We also tighten the master problem (9) with a relaxation of the subproblem. We use a very simple relaxation that requires that the processing times of tasks assigend to resource i fit between the earliest release time and the lastest deadline:

$$\sum_j p_{ij}\delta_{ij} \leq \max_j\{U_j\} - \min_j\{L_j\}$$

The subproblem is a feasibility problem because we are minimizing assignment cost only. An optimal solution is obtained as soon as the assignment is feasible on all resources. As a result, the Benders method yields no upper bound on the optimal value until it terminates with an optimal solution. This is not the case with other common objectives, such as minimizing makespan, minimizing the number of late tasks, or minimizing total tardiness. These objectives result in subproblems that are optimization problems, and they require different (and more complex) Benders cuts and subproblem relaxations. They are discussed in Hooker (2005a; 2007c).

Computational Results

The problem instances are those used in Hooker (2004; 2005a; 2007c) and elsewhere. The instances and full documentation are available at

http://web.tepper.cmu.edu/jnh/instances.htm

They consist of "c" instances, which schedule 10 to 32 tasks on 2 to 4 resources, and "e" instances, which schedule 10 to 50 tasks on 2 to 10 resources.

The "c" instances are designed to be difficult for LBBD, because the processing times on different resouces differ by as much as a factor of m when there are m resources. This causes many more tasks to be assigned to the more efficient resources, which results in a computational bottleneck on those resources. There are also fewer resources, resulting in less decoupling of the subproblem. The "e" instances are more suited to LBBD, even though they are larger. Processing times differ by at most a factor of 1.5 across resources, so that the load is more evenly (and perhaps more realistically) balanced.

We implemented LBBD by solving the master problem with CPLEX 12.4.01 and the subproblems with IBM CP Optimizer 12.4.01 using extended filtering, DFS search, and default variable and value selection. We also solved the instances by pure MIP, using the best known formulation and CPLEX 12.4.01. We did not solve the instances with pure CP, because previous experience indicates that CP solvers are considerably slower than MIP on these instances (Hooker 2005a; 2007c). All tests are run on an Intel Xeon E5345 2.33 GHz (64 bits) in single core mode with 8 GB RAM.

Table 1 shows our results for the "c" instances. LBBD is clearly superior to MIP, running at least 1000 times faster on the larger instances with more than 2 resources. The advantage is less pronounced when there are only 2 resources, which is not surprising because there is less decoupling of the subproblem. The superiority of LBBD is equally evident in the performance profile of Fig. 1.

Table 1: Computational results for the "c" instances, which are designed to be difficult for logic-based Benders decomposition (LBBD). n tasks are scheduled on m resources. The number of problem instances solved (out of 5) and average computation time in seconds are shown. Results are shown for the CPLEX MIP solver, LBBD with strengthened cuts but no subproblem relaxation in the master problem, LBBD with a relaxation and simple nogood cuts, and LBBD with a relaxation and strengthened cuts.

Size		MIP (CPLEX)		LBBD: strong cuts only		LBBD: relax + weak cuts		LBBD: relax + strong cuts	
m	n	Solved	Sec	Solved	Sec	Solved	Sec	Solved	Sec
2	10	5	0.1	5	0.2	5	0.1	5	0.1
	12	5	0.2	5	0.2	5	0.1	5	0.0
	14	5	0.1	5	0.4	5	0.1	5	0.0
	16	5	28	5	2.0	5	0.2	5	0.3
	18	5	388	5	19	5	0.5	5	0.7
	20	4	1899	5	120	5	2.0	5	8.0
	22	3	3844+	4	1852+	5	617	5	955
	24	2	4346+	1	6341+	4	1495+	4	1936+
	26	1	6362+	0	-	5	327	4	1642+
	28	2	4384+	0	-	5	1004	5	1133
	30	0	-	0	-	2	5391+	2	5761+
	32	1	5813+	0	-	2	4325+	2	4325+
3	10	5	0.0	5	0.2	5	0.1	5	0.1
	12	5	0.1	5	0.4	5	0.5	5	0.1
	14	5	0.3	5	1.2	5	0.3	5	0.2
	16	5	13	5	5.6	5	2.7	5	0.8
	18	5	548	5	22	5	7.8	5	1.4
	20	4	1712+	5	30	5	1.2	5	0.5
	22	3	3674+	5	59	5	7.5	5	2.6
	24	2	4411+	4	1739+	5	15	5	5.7
	26	0	-	4	3510+	5	191	5	98
	28	2	5238+	2	6645+	5	270	5	209
	30	0	-	0	-	4	2354+	4	1856+
	32	0	-	0	-	2	4667+	2	4751+
4	10	5	0.0	5	0.1	5	0.0	5	0.0
	12	5	0.1	5	0.2	5	0.1	5	0.1
	14	5	0.3	5	0.6	5	1.0	5	0.3
	16	5	1.0	5	0.6	5	0.4	5	0.1
	18	5	36	5	4.0	5	1.7	5	0.4
	20	5	523	5	11	5	1.1	5	0.3
	22	5	811	5	75	5	8.2	5	1.1
	24	1	6292+	5	122	5	23	5	9.1
	26	0	-	3	3369+	5	19	5	7.4
	28	1	5762+	3	4623+	5	36	5	11
	30	0	-	2	4841+	5	430	5	61
	32	0	-	0	-	5	680	5	478

+ Computation terminated after 7200 sec for instances not solved to optimality. In cases where CPLEX terminated prematurely due to lack of memory, the computation time was set at 7200 sec when computing the average time for CPLEX.

The superiority of LBBD is even clearer for the "e" instances, as revealed in Table 2 and the performance profile in Fig. 2.

The importance of the subproblem relaxation is equally evident in the results. If strengthened nogood cuts are used without a relaxation, the advantage of LBBD is substantially reduced, and it disappears completely in some instances.

Table 2: Computational results for the "e" instances.

Size		MIP (CPLEX)		LBBD: relax + weak cuts		LBBD: relax + strong cuts	
m	n	Solved	Sec	Solved	Sec	Solved	Sec
2	10	5	0.1	5	0.1	5	0.1
2	12	5	0.3	5	0.3	5	0.1
3	15	5	0.9	5	0.4	5	0.2
4	20	5	46	5	14	5	1.9
5	25	5	73	5	1.0	5	0.7
6	30	5	543	5	1.3	5	0.4
7	35	2	5122+	5	36	5	2.7
8	40	1	7246+	4	1527+	5	80
9	45	0	-	5	1050	5	35
10	50	1	6983+	5	45	5	5.4

+ Computation terminated after 7200 sec for instances not solved to optimality.

Strengthening of the nogood cuts, however, is not always effective. It actually worsens performance in "c" instances with 2 resources, and it brings rather limited improvement in the overall performance profile. Even when the instances with 2 resources are removed from the profile, as in Fig. 3, the effect of strengthening is not as great as one might expect. On the other hand, cut strengthening is significantly more effective in the "e" instances, as is evident in Fig. 2.

Analysis of Results

More detailed computational data can shed light on the results described above. Tables 3 and 4 show the average number of iterations for each instance size, as well as a breakdown of the average solution times by master problem and subproblem.

We can immediately see the effect of the subproblem relaxation. The lack of a relaxation results in a dramatic increase in the number of iterations and therefore in the solution time. This is intuitively reasonable, because a Benders method in effect computes the projection of the feasible set onto the master problem variables. As more Benders cuts are added, the projection is more accurately described. Eventually, enough cuts are added to describe the projection completely, and the problem can be solved by solving only the master problem. It is impractical to generate so many cuts, however, and the effectiveness of the Benders method rests on the fact that cut generation is guided by interim solutions of the master problem. Cuts tend to be generated only in the vicinity of the optimal solution, and this is enough to solve the problem. A relaxation of the subproblem is useful because it more tightly circumscribes the projection from the start, so that many fewer cuts are necessary to chip away regions near the optimum that are not part of the projection.

The absence of a relaxation not only generates more iterations, but each iteration requires longer to solve. This is because more iterations produce more Benders cuts, which makes the master problem larger and harder to solve. The tables suggest that a successful application of the Benders method tends to result in well under 100 iterations. After

Table 3: Computational analysis of logic-based Benders decomposition for "c" instances, showing the average number of iterations and the average computation time spent solving master problems and subproblems.

		Strong cuts only			Relax + weak cuts			Relax + strong cuts		
		Iters	Master	Subpr	Iters	Master	Subpr	Iters	Master	Subpr
m	n		sec	sec		sec	sec		sec	sec
2	10	18	0.1	0.1	9.8	0.1	0.0	4.8	0.0	0.0
	12	13	0.1	0.1	5.0	0.0	0.0	3.4	0.0	0.0
	14	19	0.1	0.3	1.8	0.0	0.0	1.8	0.0	0.0
	16	41	0.5	1.5	2.0	0.0	0.2	2.0	0.0	0.3
	18	149	5.7	14	2.4	0.0	0.5	2.4	0.0	0.7
	20	107	3.5	117	3.6	0.0	2.0	2.8	0.0	8.0
	22	340+	70+	1782+	4.6	0.0	617	4.4	0.0	955
	24	327+	67+	6263+	2.0+	0.0+	1495+	1.8+	0.0+	1936+
	26	-	-	-	1.8	0.0	327	1.6+	0.0+	1642+
	28	-	-	-	2.0	0.0	1004	1.8	0.0	1133
	30	-	-	-	4.2+	0.0+	5391+	1.0+	1452+	4309+
	32	-	-	-	1.2+	0.0+	4325+	1.0+	0.0+	4325+
3	10	13	0.0	0.1	9.8	0.1	0.0	4.4	0.0	0.0
	12	23	0.2	0.2	14	0.4	0.0	6.4	0.1	0.1
	14	42	0.7	0.5	13	0.2	0.1	6.8	0.1	0.1
	16	86	4.0	1.5	40	2.5	0.2	17	0.5	0.3
	18	183	19	3.0	61	7.3	0.5	23	1.0	0.5
	20	226	23	6.4	21	0.8	0.4	8.2	0.1	0.4
	22	340	49	10	49	2.9	4.6	16	0.4	2.3
	24	1222+	1689+	50+	55	12	3.5	22	1.6	4.1
	26	1854+	2723+	786+	130	33	158	22	0.6	97
	28	2113+	3283+	3363+	15	0.2	270	8.0	0.1	209
	30	-	-	-	80+	9.2+	2344+	21+	1.1+	1855+
	32	-	-	-	143+	64+	4602+	23+	1.7+	4750+
4	10	6.8	0.0	0.1	4.6	0.0	0.0	3.0	0.0	0.0
	12	12	0.1	0.1	6.2	0.1	0.0	4.4	0.0	0.0
	14	26	0.3	0.3	22	0.9	0.1	9.0	0.2	0.1
	16	27	0.2	0.3	12	0.3	0.1	5.6	0.1	0.1
	18	74	3.0	1.0	32	1.5	0.1	15	0.3	0.2
	20	130	9.0	2.3	26	1.0	0.2	11	0.1	0.2
	22	334	69	6.6	51	7.6	0.6	15	0.7	0.5
	24	407	104	18	96	20	3.1	37	3.4	5.6
	26	1351+	3315+	54+	83	11	7.5	32	2.0	5.4
	28	2042+	4091+	532+	27	1.7	34	12	0.5	11
	30	1408+	4665+	175+	117	395	35	41	41	20
	32	-	-	-	60	6.3	673	14	0.4	478

+Computation terminated for unsolved instances after the total computation time reaches 7200 sec.

this point, solution of the master bogs down, and the method tends to fail.

The results also reveal why the "c" instances are harder for LBBD. Once the number of tasks reaches a certain point, the subproblem solution time explodes, even while the master problem solution time remains small. This point is reached earlier when there are fewer resources, indicating that the key factor is the average number of tasks per resource. In fact, the break point is about 10 tasks per resource. Due to the uneven loads, this places subtantially more than 10 tasks on the fastest resource, whereupon the CP problem becomes highly combinatorial, and solution time explodes. This phenomenon does not occur in the "e" instances, where there are only 5 tasks per resource, and the resource loads are more evenly balanced.

Table 4: Computational analysis of logic-based Benders decomposition for "e" instances.

		Relax + weak cuts			Relax + strong cuts		
		Iters	Master	Subpr	Iters	Master	Subpr
m	n		sec	sec		sec	sec
2	10	9.4	0.1	0.0	5.2	0.0	0.0
2	12	13	0.3	0.0	4.4	0.0	0.0
3	15	14	0.4	0.0	5.6	0.1	0.1
4	20	55	14	0.0	16	1.7	0.3
5	25	19	0.4	0.0	8.6	0.1	0.6
5	30	26	1.1	0.0	8.8	0.2	0.2
7	35	76	34	0.0	19	2.0	0.7
8	40	107+	1525+	0.0+	31	78	2.1
9	45	132	1048	0.0	39	33	2.2
10	50	39	43	0.0	18	3.6	1.7

+Computation terminated for unsolved instances after the total computation time reaches 7200 sec.

From a broader perspective, the "c" instances are less suitable for LBBD because the assignment problem is relatively easy when there are only 2 or 3 resources. Most of the combinatorial complexity is relegated to the scheduling subproblem, which can quickly become intractable as the instances scale up. In the "e" instances, the assignment problem bears a more equal share of the combinatorial burden. Decoposition is more effective because the labor is more equally shared between master and subproblem.

It is less obvious how the effect of cut strengthening can be explained, but a pattern is visible. We first observe that strong cuts tend to result in greater subproblem solution time per iteration, because the subproblem is solved repeatedly in each iteration to tighten the cuts. However, this tends to be more than offset by the smaller number of iterations. As one might expect, stronger cuts remove more solutions that lie outside the projection, and fewer cuts are therefore necessary. The key observation is that when there are fewer resources relative to tasks, the subproblem relaxation already substantially reduces the number of iterations. This means that there is less room for further reduction due to strong cuts. This is particularly evident in the "c" instances with only 2 or 3 resources (Table 3).

The data further suggest that cut generation should result in a rough balance between the total master solution time and the total subproblem solution time. This is particularly evident for the "e" instances (Table 4), for which weak cuts result in nearly zero subproblem solution time, while strong cuts roughly equalize the master and subproblem time in most cases. LBBD is somewhat less efficient for instances with 8 and 9 resources, for which the master solution time is significantly greater than the subproblem time. This suggests that one should invest more time to generate stronger cuts for these instances.

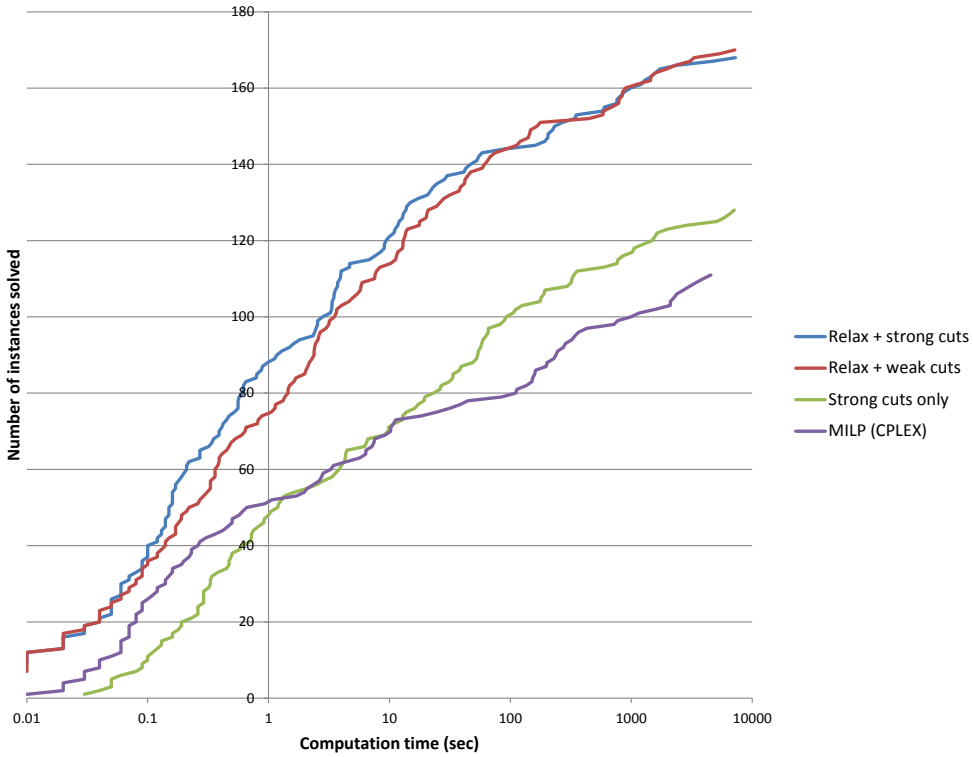

Figure 1: Performance profile for all 180 "c" instances.

Conclusions

Based on the experiments described above, we conclude that logic-based Benders decomposition is more effective when the problem in fact decomposes. The planning and scheduling components should both embody substantial shares of the problem's combinatorial complexity.

When the planning portion involves the assigment of tasks to resources, the average number of tasks per resource should be small enough that the assignment problem is roughly as hard as the scheduling problem. In particular, it should remain below the threshold at which solution time of the scheduling problem tends to explode. LBBD failure most often occurs when the CP solver blows up because of too many tasks are assigned to the resource. This is less likely to occur when no resource is significantly faster than others.

We also find that the most effective technique for reducing solution time is to include a relaxation of the subproblem in the master problem. This can dramatically reduce the number of iterations, as well as the solution time per iteration, because the master problems are smaller when there are fewer iterations.

The identification of stronger cuts can also significantly reduce the number of iterations, especially when the master and subproblem complexity are properly balanced. A rough guideline is to invest greater time in identifying strong cuts when the subproblem solution time per Benders iteration is less than the master problem solution time.

In any event, LBBD remains substantially faster than state-of-the art mixed integer programming on the problem class studied here. This is despite the marked improvement of MIP solvers, already highly advanced, over the last few years. The advantage exceeds a factor of 1000 for larger instances. In fact, the LBBD method presented here is best conceived as an enchancement of MIP rather than a competitor, because MIP solves the master problem, and LBBD will therefore improve as MIP improves. LBBD also benefits from advancements in CP, used here to solve the subproblem.

On the other hand, LBBD is likely to fail when the imbalance or size of the master and subproblem result in more than 100 Benders iterations.

These results suggest several research directions. When the number of tasks per resource crosses the CP solver's tractability threshold, one option is to solve the scheduling subproblem with failure-directed search, which was recently introduced in the IBM CP optimizer. Another option is to decompose the scheduling problem itself using LBBD. This can be done by dividing the time horizon into segments and creating a subproblem for each segment. This approach has been successfully applied in the context of single-resource scheduling (Çoban and Hooker 2013).

The overriding importance of a subproblem relaxation suggests that further research should be conducted in this area. For example, some of the subproblem constraints could be incorporated verbatim into the master problem,

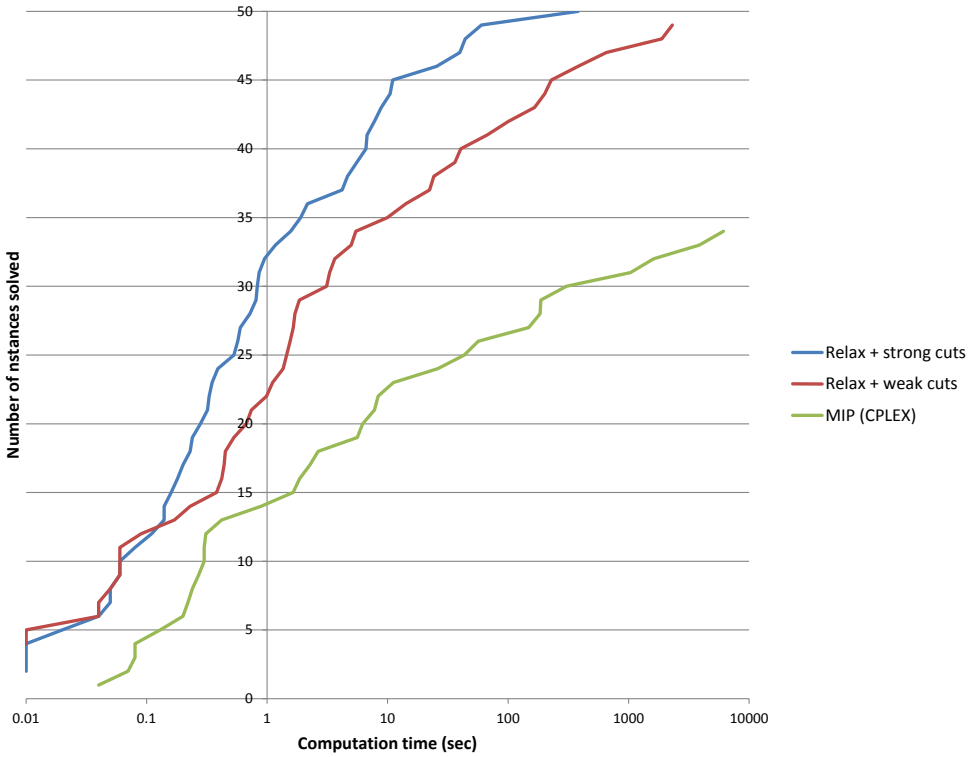

Figure 2: Performance profile for all 50 "e" instances.

along with the subproblem variables that occur within them. Normally, subproblem variables do not occur in the master, but their inclusion is consistent with the Benders mechanism so long as their solution values in the master are discarded before the next subproblem is solved. Their presence in the master serves only to restrict the possible values of the master variables. This strategy has been applied in a Benders-based solution of the home health care delivery problem (Ciré and Hooker 2012). A practice of decomposing problems while allowing the components to intermingle could lead to a research program that improves decomposition methods in general.

One characteristic of LBBD not exploited here is that the subproblem is easily parallelized by assignining each facility to a different processor. This could be a significant advantage if there are a large number of facilities, or if the scheduling subproblem is itself decomposed.

Finally, the generation of strong Benders cuts could benefit from access to dual information in the subproblem solver. That is, the solver should report how it proved optimality or infeasiblity. The cuts used here were strengthened by reperatedly re-solving the subproblem to tease out dual information, but cuts based on the actual dual solution could be much more effective.

References

Baptiste, P.; Pape, C. L.; and Nuijten, W. 2001. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. Dordrecht: Kluwer.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4:238–252.

Benini, L.; Bertozzi, D.; Guerri, A.; and Milano, M. 2005. Allocation and scheduling for MPSoCs via decomposition and no-good generation. In *Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, 107–121. Springer.

Cambazard, H.; Hladik, P.-E.; Déplanche, A.-M.; Jussien, N.; and Trinquet, Y. 2004. Decomposition and learning for a hard real time task allocation problem. In Wallace, M., ed., *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, 153–167. Springer.

Çoban, E., and Hooker, J. N. 2013. Single-facility scheduling by logic-based benders decomposition. *Annals of Operations Research* 210:245–272.

Chu, Y., and Xia, Q. 2004. Generating Benders cuts for a class of integer programming problems. In Régin, J. C., and Rueher, M., eds., *CPAIOR 2004 Proceedings*, volume 3011 of *Lecture Notes in Computer Science*, 127–141. Springer.

Ciré, A., and Hooker, J. N. 2012. A heuristic logic-based benders method for the home health care problem. Presented at Matheuristics 2012, Angra dos Reis, Brazil.

Ciré, A. A.; Çoban, E.; and Hooker, J. N. 2013. Mixed integer programming vs logic-based Benders decomposition

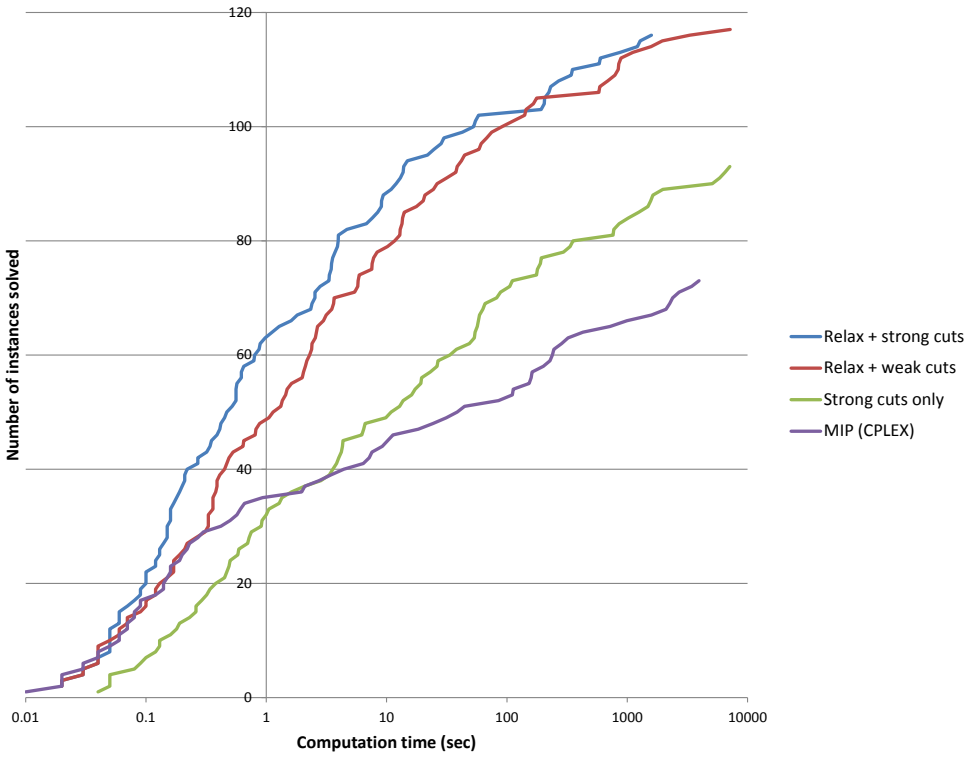

Figure 3: Performance profile for the 120 "c" instances with 3 and 4 resources.

for planning and scheduling. In Gomes, C., and Sellmann, M., eds., *CPAIOR 2013 Proceedings*, 325–331.

Corréa, A. I.; Langevin, A.; and Rousseau, L. M. 2004. Dispatching and conflict-free routing of automated guided vehicles: A hybrid approach combining constraint programming and mixed integer programming. In Régin, J. C., and Rueher, M., eds., *CPAIOR 2004 Proceedings*, volume 3011 of *Lecture Notes in Computer Science*, 370–378. Springer.

Fazel-Zarandi, M. M., and Beck, J. C. 2009. Solving a location-allocation problem with logic-based Benders decomposition. In Gent, I. P., ed., *Principles and Practice of Constraint Programming (CP 2009)*, volume 5732 of *Lecture Notes in Computer Science*, 344–351. New York: Springer.

Geoffrion, A. M. 1972. Generalized Benders decomposition. *Journal of Optimization Theory and Applications* 10:237–260.

Harjunkoski, I., and Grossmann, I. E. 2001. A decomposition approach for the scheduling of a steel plant production. *Computers and Chemical Engineering* 25:1647–1660.

Harjunkoski, I., and Grossmann, I. E. 2002. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers and Chemical Engineering* 26:1533–1552.

Hooker, J. N., and Ottosson, G. 2003. Logic-based Benders decomposition. *Mathematical Programming* 96:33–60.

Hooker, J. N., and Yan, H. 1995. Logic circuit verification by Benders decomposition. In Saraswat, V., and Hentenryck, P. V., eds., *Principles and Practice of Constraint Programming: The Newport Papers*, 267–288. Cambridge, MA: MIT Press.

Hooker, J. N. 1995. Logic-based Benders decomposition. In *INFORMS National Meeting (INFORMS 1995)*.

Hooker, J. N. 1996. Inference duality as a basis for sensitivity analysis. In Freuder, E. C., ed., *Principles and Practice of Constraint Programming (CP 1996)*, volume 1118 of *Lecture Notes in Computer Science*, 224–236. Springer.

Hooker, J. N. 2000. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. New York: Wiley.

Hooker, J. N. 2004. A hybrid method for planning and scheduling. In Wallace, M., ed., *Principles and Practice of Constraint Programming (CP 2004)*, volume 3258 of *Lecture Notes in Computer Science*, 305–316. Springer.

Hooker, J. N. 2005a. A hybrid method for planning and scheduling. *Constraints* 10:385–401.

Hooker, J. N. 2005b. Planning and scheduling to minimize tardiness. In *Principles and Practice of Constraint Programming (CP 2005)*, volume 3709 of *Lecture Notes in Computer Science*, 314–327. Springer.

Hooker, J. N. 2006. An integrated method for planning and scheduling to minimize tardiness. *Constraints* 11:139–157.

Hooker, J. N. 2007a. *Integrated Methods for Optimization*. Springer.

Hooker, J. N. 2007b. Planning and scheduling by logic-based Benders decomposition. *Operations Research* 55:588–602.

Hooker, J. N. 2007c. Planning and scheduling by logic-based Benders decomposition. *Operations Research* 55:588–602.

Hooker, J. N. 2012. *Integrated Methods for Optimization, 2nd ed.* Springer.

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

Maravelias, C. T., and Grossmann, I. E. 2004. Using MILP and CP for the scheduling of batch chemical processes. In Régin, J. C., and Rueher, M., eds., *CPAIOR 2004 Proceedings*, volume 3011 of *Lecture Notes in Computer Science*, 1–20. Springer.

Terekhov, D.; Beck, J. C.; and Brown, K. N. 2007. Solving a stochastic queueing design and control problem with constraint programming. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, volume 1, 261–266. AAAI Press.

Thorsteinsson, E. 2001. Branch and check: A hybrid framework integrating mixed integer programming and constraint logic programming. In Walsh, T., ed., *Principles and Practice of Constraint Programming (CP 2001)*, volume 2239 of *Lecture Notes in Computer Science*, 16–30. Springer.

Timpe, C. 2002. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum* 24:431–448.

Yunes, T. H.; Aron, I.; and Hooker, J. N. 2010. An integrated solver for optimization problems. *Operations Research* 58:342–356.

A Constraint-based Optimizer for Scheduling Solar Array Operations on the International Space Station

Jan Jelínek and Roman Barták
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha, Czech Republic

Abstract

Space applications are demanding for complex operation and safety constraints and contain many objectives. This demand is even stronger for human spaceflight missions such as the International Space Station (ISS). This paper describes a novel approach to plan and schedule solar array operations on the ISS in a safe and effective manner. Opposite to previous approaches, it assumes global optimization while still taking in account safety and operation constraints. The paper proposes a constraint model to describe the problem formally and discusses methods to solve the model.

Introduction

Automated planning and scheduling deal with the problem of deciding which activities are necessary to reach the goal (planning) and when and where these activities should be executed (scheduling). These two tasks can be decoupled but if planning and scheduling are closely interconnected then it is more appropriate to solve both tasks together. In this paper we present an integrated constraint-based planner and scheduler to generate schedules for solar array operations on the International Space Station (ISS).

Solar arrays at the ISS are designed to automatically track the sun, which is what they do most of the time. However, some ISS operations such as docking a spacecraft, extra vehicular activities, water dumps, thruster firings etc. impose additional constraints on solar array operations to prevent thermal stresses, environmental contamination, and structural loads. In such situations, solar arrays may need to be parked or even locked/latched, which must be planned in advance according to expected operations of the ISS.

Currently, the solar array planning problem is solved manually by a team of people known as PHALCONs (Power, Heating, and Lighting Controllers). It takes about four weeks to manually produce an ISS solar array operations plan for a typical four-week planning horizon. The Solar Array Constraint Engine (SACE) was proposed to automatically generate solar array operations plans subject to all operation constraints and user-configurable solution preferences (Reddy *et al.,* 2011). The SACE uses an approach similar to manual scheduling with left-to-right greedy scheduling. The advantage is tractability of sub-problems solved, but in principle the schedule is suboptimal and may not be found at all even if a feasible plan exists.

In this paper we propose a constraint-based optimizer for scheduling solar array operations (COSSA) that does global optimization. In particular, we formulate the problem as a constraint satisfaction problem where the planning component is modeled using optional activities with possibly zero durations. The problem formulation is taken mainly from the challenge domain at the International Competition on Knowledge Engineering for Planning and Scheduling 2012 (Frank, 2012). Our approach and the SACE are the only two automated planners for this domain.

We first introduce the solar array operations planning domain and highlight the properties of the SACE approach. Then we describe our method in detail and discuss suggested solving techniques for the constraint model. Finally, we experimentally evaluate the proposed model and compare it with our implementation of the SACE method.

The Problem and Existing Approaches

This section sketches the main parts of the problem solved; full details can be found in (Frank, 2012; Reddy *et al.*, 2011).

The ISS has eight solar arrays, each of which is mounted on a rotary joint called the Beta Gimbal Assembly (BGA). The solar arrays are split into two groups each of which consists of four solar arrays mounted via the Solar Array Rotary Joint (SARJ) to the station (Figure 1). Thus each panel has two degrees of rotational freedom, though one degree of freedom is shared between the panels in the same group. Each rotary joint can be in exactly one mode: Autotrack, Park, or Lock (Latch for BGA), or the joint can turn between the modes. The state of each joint is also described by the angle of orientation (360 positions).

In the Autotrack mode, the onboard software automatically rotates the panel so its surface is pointing directly onto the sun to maximize energy generated. In the formal model a known constant speed of rotation is assumed for this mode. The Autotrack mode must last at least 90 minutes. In the Park mode, a drive motor is engaged to maintain the current array angle, while in the Lock and Latch modes, a physical barrier is engaged. Transition into and out of Lock/Latch modes takes 20 minutes.

The input to the solar array planning problem consists of a sequence of configurations, where each configuration starts

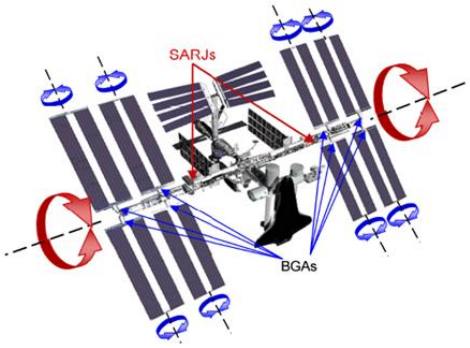

Figure 1: Solar arrays connected via rotary joints to ISS (taken from (Frank, 2012)).

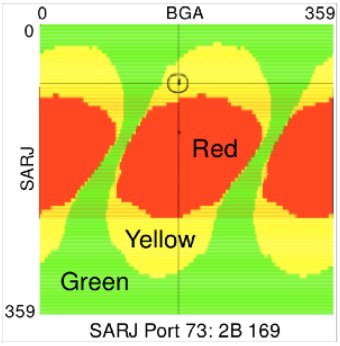

Figure 2: A table indicating, for one SARJ and one BGA, the safety areas for rotation angles (taken from (Frank, 2012)).

when the previous configuration finishes. This time may be flexible and in such a case the planner decides the appropriate time. This is the only situation when the groups of solar arrays interact; otherwise they can be scheduled independently. Each panel can be in exactly one mode in each configuration. The configuration also determines whether turning is disallowed (docking, undocking, reboost, maneuver), allowed at the end of the configuration (approach, prop purge, water dump), or allowed both at the beginning and at the end (attitude hold). It also defines a maximum rotation speed for SARJs (it is fixed for BGAs) and a contingency mode when some constraints can be violated. Finally, there are other parameters of the configuration determining for a pair of BGA and SARJ a set of four soft constraints: Power Generation (P), Structural Load (L), Environmental Contamination (E), and Longeron Shadowing (S). Each of these constraints is expressed as a 360×360 table (Figure 2) with three types of values: Green (preferred/best), Yellow (acceptable), and Red (infeasible in most situations/worst). The table is used as follows. If both BGA and SARJ are parked or locked/latched at some orientations then the value of the constraint is at the intersection of row and column corresponding to the orientations. If BGA (SARJ) is autotracking and SARJ (BGA) is parked or locked then the value of the constraint is the worst value in a row (column) defined by the orientation of SARJ (BGA). If both BGA and SARJ are autotracking then the value of the constraint is the worst value in the whole table. It is not allowed to use orientations with the red value for P, L, and S tables. Red value is allowed for the E table, but it is reflected in the quality of the plan. Turning of BGA can start only after turning of its SARJ finished and all BGAs (for a given SARJ) must start turning at the same time. The above tables are not assumed during turning.

The task is to determine for each joint in each configuration the following sequence of "activities":

- unlocking – transition out of lock/latch (optional)
- turning to the required orientation (optional)
- being in a selected mode
- turning to the next required orientation (optional)
- locking – transition into lock/latch (optional)

There might be some wait times between the above "activities", for example BGA turning waits until SARJ turning finishes. A joint can be in a selected mode in several consecutive configurations to satisfy the minimal duration constraint of the mode.

The plans are evaluated using four criteria. The most important criterion is color in the tables: table S first, then tables L and E, and finally table P (Figure 3). The second criterion is mode: Autotrack is preferred, followed by Park, and Lock/Latch to be last. Then, the number of changes in rotations should be minimized (the joint can rotate in positive and negative directions). Finally, the time spent in turning should be minimized. We are looking for a schedule such that no configuration can be scheduled better without worsening the schedule of another configuration. However, it is not necessary to guarantee the optimality of the found schedule.

SACE

The Solar Array Constraint Engine (SACE) is an automated planner for the solar array problem that mimics a human approach (Reddy *et al.*, 2011). The SACE uses left-to-right scheduling. It decides the best orientation of joints and best modes in a given configuration (going from left to right). If this plan is not compatible with the plan from the previous configuration (not enough time for transition) then both configurations are merged and a new plan for the merged configuration is looked for. This also assumes "worst-case" merge of the table constraints. Then look-ahead is used to prune infeasible modes for the next configuration. The process is repeated until a plan for all configurations is found (no backtracking to previous configurations is allowed).

The system is based on constraint solving, namely the SACE is built on top of the AI planning system EUROPA (Frank and Jónsson, 2003). It exploits the tree structure of the constraint network to find a plan for a single configuration but due to the configuration merging, the schedule is suboptimal, and the algorithm could fail to find a valid plan even if one exists. This is the reason why we propose another constraint-based approach without these deficiencies.

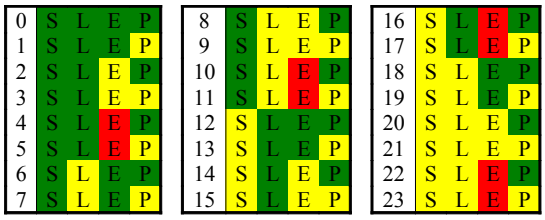

Figure 3: Color preferences order.

Constraint Model

This section describes the model used for solar array planing. First we define necessary variables and constants; then, we describe an objective function and, finally, we focus on different sets of constraints.

Variables and Their Domains

First, we describe the representation of the output plan. For each joint j on the side s in the configuration i we introduce the following variables. $M_{i,s,j} \in \{AUTOTRACK, PARK, LOCK\}$ is a variable describing the mode of the joint. Each joint is in exactly one orientation (angle) during modes $PARK$ and $LOCK$ which is described by variable $O_{i,s,j} \in [0; 360)$ for the orientation of the joint. During the mode $AUTOTRACK$ the orientation changes continuously, so $O_{i,s,j}$ represents the orientation at the end of the mode before its final turning. Each joint can turn at the beginning or at the end of each configuration, so we have variables $R^B_{i,s,j}, R^E_{i,s,j} \in (-360; 360)$ for the angle and direction of the turning, where $R^B_{i,s,j}$ represents the turning angle at the beginning and $R^E_{i,s,j}$ represents the turning angle at the end of the configuration. Variable T_i represents the start time of the configuration i in minutes (also the end time of the configuration $i-1$).

The input defines the SARJ turn rate $S_i \in [9;\ 30]$ in degrees per minute and the type of the event (whether it is disallowed to turn joints, or to turn joints only at the end, or to turn joints also at the beginning of the configuration) $F_i \in \{NO; END; BOTH\}$ for each configuration i. Both S_i and F_i are constants. The input defines domains of variables T_i (see above). The last part of the input is the set of color loss functions $c_{i,s,b}(M_{i,s,a}, M_{i,s,b}, O_{i,s,a}, O_{i,s,b}) \rightarrow [0;\ 23]$ that for each feasible combination of orientations and modes of BGA b and corresponding SARJ a on the side s assigns a score of its color in the configuration i. This function combines tables P, L, E, S (Figure 2) with a given color preference order (Figure 3). It is represented as a table constraint (Carlsson *et al.*, 1997) defining the quality of schedule.

For each configuration i and side s we used the following auxiliary variables for the formulation of constraints. $D^U_{i,s,j}$ denotes how long the joint j must unlock at the beginning of the configuration. Similarly, $D^L_{i,s,j}$ denotes the time of locking at the end of the configuration. Both unlocking and locking take 20 minutes, but the remaining time can propagate from one configuration to the adjacent configuration if the first one is too short. So domains of $D^U_{i,s,j}$ and $D^L_{i,s,j}$ are $[0; 20]$. $D^B_{i,s,j}$ denotes how long the joint j turns at the beginning of the configuration. Similarly, $D^E_{i,s,j}$ denotes the time of the turning at the end of the configuration. For each BGA b, $D^B_{i,s,b} = \lceil R^B_{i,s,b}/18 \rceil$ and $D^E_{i,s,b} = \lceil R^E_{i,s,b}/18 \rceil$, because the BGA turn rate is 18 °/min. So domains of $D^B_{i,s,b}$ and $D^E_{i,s,b}$ are $[0; 20]$ for all BGAs. For SARJ a, $D^B_{i,s,a} = \lceil R^B_{i,s,a}/S_i \rceil$ and $D^E_{i,s,a} = \lceil R^E_{i,s,a}/S_i \rceil$ depends on the SARJ turn rate S_i during the configuration so their domains are $[0; \lceil 360/S_i \rceil]$. $D^S_{i,s,j}$ denotes how long the joint j cannot turn at the beginning of the configuration. For the SARJ a, $D^S_{i,s,a}$ depends on the $D^U_{i,s,a}$ so its domain is $[0; 20]$. For the BGA b, $D^S_{i,s,b}$ depends also on the $D^S_{i,s,a}$ of the corresponding SARJ a so its domain is $[0; 20 + \lceil 360/S_i \rceil]$. $D^F_{i,s,j}$ denotes when the joint j must begin with the final turning relatively to the end of the configuration. For the BGA b, $D^F_{i,s,b}$ depends on the $D^E_{i,s,b}$ and $D^L_{i,s,b}$ so its domain is $[0; 40]$. For the SARJ a, $D^F_{i,s,a}$ depends on the $D^S_{i,s,a}$ and $D^F_{i,s,b}$ for all BGAs b, so its domain is $[0; 40 + \lceil 360/S_i \rceil]$. $D^A_{i,s,j}$ denotes how long the joint i is in the Autotrack mode during the configuration, so its domain is $[0; \infty]$. The last variable is $D^C_{i,s,j}$ that denotes how long the joint i was in the uninterrupted Autotrack mode until the end of the configuration, so its domain is $[0; \infty]$.

Time dependency of variables is shown in Figure 4.

Objective Function

There is no single global objective function given, there are only partial objective functions $f_{i,s}$, one for each configuration i and side s. Each of these functions has four parts. The most important is the cost of colors:

$$f^C_{i,s} = \sum\nolimits_b c_{i,s,b}(M_{i,s,a}, M_{i,s,b}, O_{i,s,a}, O_{i,s,b}).$$

The second cost function is the cost of the modes:

$$f^M_{i,s} = \sum\nolimits_j m(M_{i,s,j}),$$

where m is the loss function for modes meeting the condition $m(AUTOTRACK) < m(PARK) < m(LOCK)$.

The third cost function is the number of changes of directions $f^D_{i,s}$ that depends on $M_{i-1,s,j}$, $R^E_{i-1,s,j}$, $R^B_{i,s,j}$, $M_{i,s,j}$ and $R^E_{i,s,j}$. There are three positions, where the change of the direction may occur:

- before the $R^B_{i,s,j}$: $(R^E_{i-1,s,j} > 0 \wedge R^B_{i,s,j} < 0)$, or $(R^E_{i-1,s,j} < 0 \wedge R^B_{i,s,j} > 0)$, or $(M_{i-1,s,j} = AUTOTRACK \wedge R^E_{i-1,s,j} = 0 \wedge R^B_{i,s,j} < 0)$;
- before the $M_{i,s,j}$: $(R^B_{i,s,j} < 0 \wedge M_{i,s,j} = AUTOTRACK)$ or $(R^E_{i-1,s,j} < 0 \wedge R^B_{i,s,j} = 0 \wedge M_{i,s,j} = AUTOTRACK)$;
- between $M_{i,s,j}$ and $R^E_{i,s,j}$: $(M_{i,s,j} = AUTOTRACK \wedge R^E_{i,s,j} < 0)$.

The least important function is the cost of turning (measured by the turning angle):

$$f^L_{i,s} = \sum\nolimits_j |R^B_{i,s,j}| + |R^E_{i,s,j}|.$$

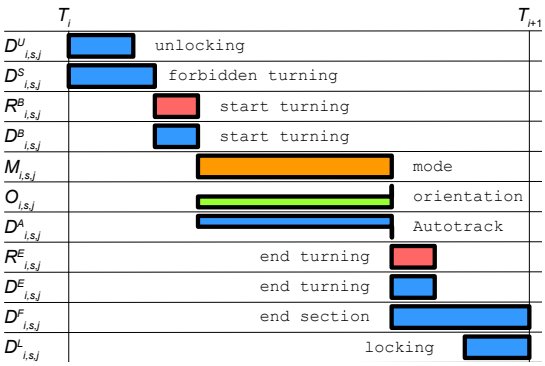

Figure 4: Time dependency of variables during a single configuration (ordering of activities within a configuration).

The optimal plan of the configuration i at the side s is the plan with the lexicographically smallest vector $(f^C_{i,s}, f^M_{i,s}, f^D_{i,s}, f^L_{i,s})$.

Locking & Unlocking Constraints

No joint can be simultaneously locked and unlocked, but the unlocking time or the locking time can be longer than the duration of the configuration. This restriction is defined using the constraint:

$$D^U_{i,s,j} + D^L_{i,s,j} \leq T_{i+1} - T_i \vee D^U_{i,s,j} = 0 \vee D^L_{i,s,j} = 0.$$

If some joint is locked during the configuration *i* and if it isn't locked in the configuration $i+1$, then it must be unlocked during the first 20 minutes of the configuration $i+1$. If the configuration i is too short for unlocking, the remaining unlocking time is propagated to the configuration $i+1$. These restrictions are represented by the set of constraints:

$$M_{i+1,s,j} = LOCK \Rightarrow D^U_{i+1,s,j} = 0,$$

$$(M_{i,s,j} = LOCK \wedge M_{i+1,s,j} \neq LOCK) \Rightarrow D^U_{i+1,s,j} = 20,$$

$$(M_{i,s,j} \neq LOCK \wedge M_{i+1,s,j} \neq LOCK) \Rightarrow$$
$$D^U_{i+1,s,j} = max\{D^U_{i,s,j} - T_{i+1} + T_i; 0\}.$$

The situation is symmetric in the case of locking and a similar set of constraints is used.

Turning Constraints

The type of event must allow turning, which is modeled as:

$$F_i = END : R^B_{i,s,j} = 0,$$

$$F_i = NO : R^B_{i,s,j} = 0 \wedge R^E_{i,s,j} = 0.$$

The joint cannot turn, if it is locked, i.e.

$$M_{i,s,j} = LOCK \Rightarrow R^B_{i,s,j} = 0 \wedge R^E_{i,s,j} = 0.$$

There must be enough time to execute turning. It is a complex rule that requires some auxiliary variables to express relationships between individual joints. For the beginning of the configuration: Turning of SARJ a cannot start earlier than after the SARJ is unlocked. Turning of any BGA b cannot start earlier than after finishing turning of corresponding SARJ a (if SARJ turns at all) and also after all BGAs are ready as they must start turning at the same time. Formally

$$D^S_{i,s,a} = D^U_{i,s,a},$$

$$R^B_{i,s,a} = 0 \implies D^S_{i,s,B} = \max\{0, \max_{b, R^B_{i,s,b} \neq 0} D^U_{i,s,b}\},$$

$$R^B_{i,s,a} \neq 0 \implies D^S_{i,s,B} = \max\{D^U_{i,s,a} + D^B_{i,s,a}, \max_{b, R^B_{i,s,b} \neq 0} D^U_{i,s,b}\}.$$

For the end of the configuration: first, we compute when all BGAs must start their turnings relatively to the end of the configuration; then we can compute when SARJ must finish its turning (before the BGAs start turning).

$$D^F_{i,s,B} = \max_{b, R^E_{i,s,b} \neq 0} (D^E_{i,s,b} + D^L_{i,s,b}),$$

$$R^E_{i,s,a} = 0 \implies D^F_{i,s,a} = D^F_{i,s,B},$$

$$R^E_{i,s,a} \neq 0 \implies D^F_{i,s,a} = \max\{D^L_{i,s,a}, D^F_{i,s,B}\} + D^E_{i,s,a}.$$

Finally, it is possible to check whether there is enough time for turning on each joint. This requirement is expressed by the following set of constraints:

$$D^U_{i,s,j} + D^L_{i,s,j} \geq T_{i+1} - T_i \implies (D^B_{i,s,j} = 0 \wedge D^E_{i,s,j} = 0),$$

$$(D^U_{i,s,j} + D^L_{i,s,j} < T_{i+1} - T_i \wedge D^B_{i,s,j} = 0 \wedge D^E_{i,s,j} > 0)$$
$$\implies D^U_{i,s,j} + D^F_{i,s,j} \leq T_{i+1} - T_i,$$

$$(D^U_{i,s,j} + D^L_{i,s,j} < T_{i+1} - T_i \wedge D^B_{i,s,j} > 0 \wedge D^E_{i,s,j} = 0)$$
$$\implies D^S_{i,s,j} + D^B_{i,s,j} + D^L_{i,s,j} \leq T_{i+1} - T_i,$$

$$(D^U_{i,s,j} + D^L_{i,s,j} < T_{i+1} - T_i \wedge D^B_{i,s,j} > 0 \wedge D^E_{i,s,j} > 0)$$
$$\implies D^S_{i,s,j} + D^B_{i,s,j} + D^F_{i,s,j} \leq T_{i+1} - T_i.$$

To strengthen domain filtering, we implemented the above implication constraints using a single *case* constraint (Carlsson *et al.*, 1997) as Figure 5 shows.

The last constraint models that every joint j on the side with SARJ a finishes turning at the beginning of the item before any joint starts turning at the end of the item:

$$\max_j \{D^S_{i,s,j} + D^B_{i,s,j}\} + D^F_{i,s,a} \leq T_{i+1} - T_i.$$

Autotrack Constraints

The duration of the Autotrack mode during one configuration must be computed to determine the orientation of the joint at the end of the configuration. This computation is similar to the check of the duration of the turning described in the previous section (Figure 5).

$$M_{i,s,j} \neq AUTOTRACK \implies D^A_{i,s,j} = 0,$$

$$(M_{i,s,j} = AUTOTRACK \wedge D^B_{i,s,j} = 0 \wedge D^E_{i,s,j} = 0)$$
$$\implies D^A_{i,s,j} = T_{i+1} - T_i - D^U_{i,s,j} - D^L_{i,s,j},$$

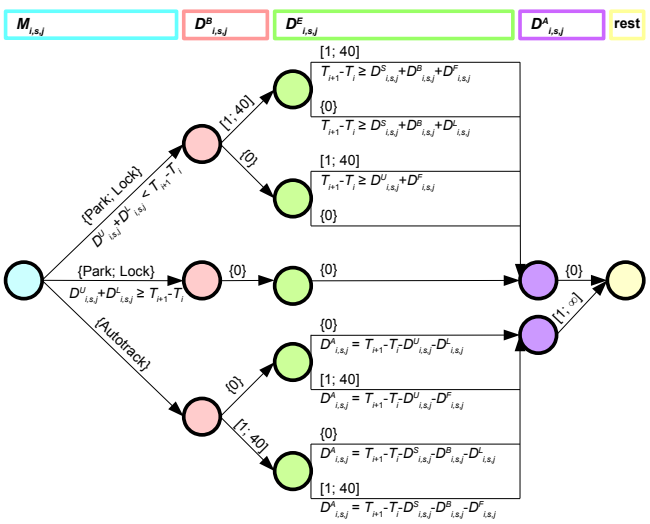

Figure 5: A graph representation of the case constraints on the duration of the configuration and computation of $D^A_{i,s,j}$. Each node corresponds to a variable defined in the header (highlighted with the same color). A directed edge restricts the domain of the variable corresponding to the starting node of that edge. Additionally, each edge can be labeled with a linear inequality constraint to be also satisfied.

$$(M_{i,s,j} = AUTOTRACK \wedge D^B_{i,s,j} = 0 \wedge D^E_{i,s,j} > 0) \implies D^A_{i,s,j} = T_{i+1} - T_i - D^U_{i,s,j} - D^F_{i,s,j},$$
$$(M_{i,s,j} = AUTOTRACK \wedge D^B_{i,s,j} > 0 \wedge D^E_{i,s,j} = 0) \implies D^A_{i,s,j} = T_{i+1} - T_i - D^S_{i,s,j} - D^B_{i,s,j} - D^L_{i,s,j},$$
$$(M_{i,s,j} = AUTOTRACK \wedge D^B_{i,s,j} > 0 \wedge D^E_{i,s,j} > 0) \implies D^A_{i,s,j} = T_{i+1} - T_i - D^S_{i,s,j} - D^B_{i,s,j} - D^F_{i,s,j}.$$

The constraint determining the orientation $O_{i,s,a}$ of the SARJ a is following:

$$O_{i,s,a} = (O_{i-1,s,a} + R^E_{i-1,s,a} + R^B_{i,s,a} + S_i \times D^A_{i,s,a}) \mod 360.$$

For the BGA b, the constraint is very similar:

$$O_{i,s,b} = (O_{i-1,s,b} + R^E_{i-1,s,b} + R^B_{i,s,b} + 18 \times D^A_{i,s,b}) \mod 360.$$

The Autotrack mode must last at least 90 minutes. This restriction can be satisfied in several consecutive configurations, i.e.

$$M_{i,s,j} \neq AUTOTRACK \implies D^C_{i,s,j} = 0,$$
$$(M_{i,s,j} = AUTOTRACK \wedge (D^U_{i,s,j} = 0 \wedge D^B_{i,s,j} = 0) \wedge (D^E_{i,s,j} = 0 \wedge D^L_{i,s,j} = 0)) \implies D^C_{i,s,j} = D^C_{i-1,s,j} + D^A_{i,s,j},$$
$$(M_{i,s,j} = AUTOTRACK \wedge (D^U_{i,s,j} = 0 \wedge D^B_{i,s,j} = 0) \wedge (D^E_{i,s,j} > 0 \vee D^L_{i,s,j} > 0)) \implies (D^C_{i-1,s,j} + D^A_{i,s,j} \geq 90 \wedge D^C_{i,s,j} = 0),$$
$$(M_{i,s,j} = AUTOTRACK \wedge (D^U_{i,s,j} > 0 \vee D^B_{i,s,j} > 0) \wedge (D^E_{i,s,j} = 0 \wedge D^L_{i,s,j} = 0)) \implies D^C_{i,s,j} = D^A_{i,s,j},$$
$$(M_{i,s,j} = AUTOTRACK \wedge (D^U_{i,s,j} > 0 \vee D^B_{i,s,j} > 0) \wedge (D^E_{i,s,j} > 0 \vee D^L_{i,s,j} > 0)) \implies (D^A_{i,s,j} \geq 90 \wedge D^C_{i,s,j} = 0).$$
$$(M_{i,s,j} = AUTOTRACK \wedge D^B_{i,s,j} = 0) \vee D^C_{i-1,s,j} = 0 \vee D^C_{i-1,s,j} \geq 90.$$

Figure 6 shows the graph for the case constraint that checks the duration of the Autotrack mode.

Optimization Algorithm

The plan for each configuration i and side s has assigned a vector of four values $(f^C_{i,s}, f^M_{i,s}, f^D_{i,s}, f^L_{i,s})$ that evaluates its quality (lexicographically). To evaluate the quality of the whole plan we need to put together all these quadruples into a single vector while respecting the priorities within each quadruple. One option is to simply concatenate these quadruples in the order of configurations, but this would prioritize the earlier configurations "too much" ($f^L_{i,s}$ would be more important than $f^C_{i+1,s}$). Hence we decided to construct the global evaluation vector by putting first the values $f^C_{i,s}$ for all configurations and sides, then the values $f^M_{i,s}$ followed by $f^D_{i,s}$ and finally $f^L_{i,s}$. We have chosen this option, because the achievable color depends on the duration of turning nonlinearly and because a small change in the duration of turning can result in a significant difference in the achievable color. This approach should result in less stress on solar panels.

Now we describe, how the multi-criteria optimization is realized to get the lexicographically best solution. First we find an optimal solution (assignment of all variables) for the first objective function. Then we solve the problem again, but with an extra constraint binding the value of the first objective function to the just computed optimum and optimizing the second objective function etc. until all functions are optimized. This way we find the lexicographically best solution. So for every extended problem at this level of abstraction there exists a solution, if the first search iteration was successful. Consequently it is possible to interrupt the algorithm at any time and get at least a partially optimized plan, if it isn't enough time for full optimization.

Optimization of a Single Objective

Branch and bound method (Land and Doig, 1960) is used to find an optimal solution for a given objective function. First, any solution is found. Then the algorithm is looking for another solution that is better than the last found solution until this extended problem has no solution or until the lower bound of that function is reached. It is possible to use suboptimal solutions if finding the optimal solution (or proving its optimality) is too computationally expensive. Therefore all iterations are running with a time limit.

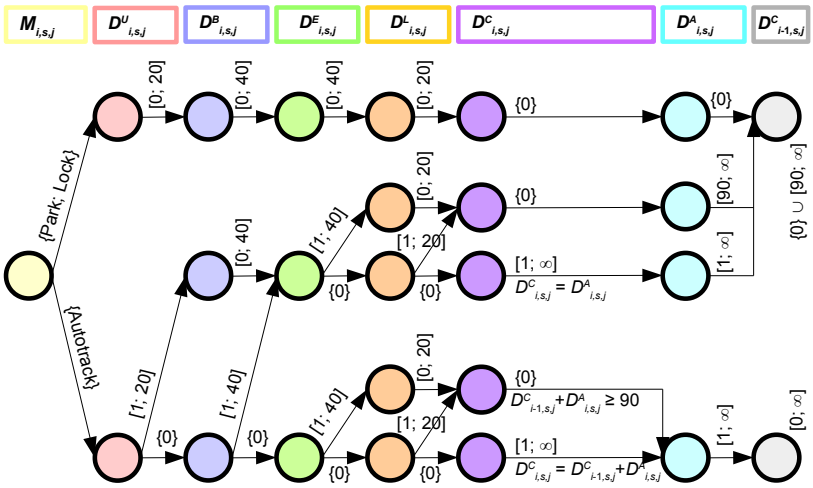

Figure 6: A graph representation of the case constraint on the duration of the Autotrack mode.

When searching for the solution, classical methods for solving CSPs are used, that is, labeling of variables via depth-first search with maintaining consistency. Various variable ordering heuristics and branching schemes can be used during labeling. Different strategies can have different success rates and runtimes in different situations. Therefore we don't use one fixed strategy, but we exploit a set of strategies.

During optimization of each objective function (except the first one) the value of the current objective function from the last solution found during optimization of the previous objective function is used as the first upper bound of the current objective function. That is possible because no infeasible constraints have been added – in the worst case we prove optimality of the previous solution according to the current objective function

When an objective function is optimized for the first time, no search is necessary if its upper bound is equal to its lower bound. This saves time, because the evaluation of the function on the known plan is faster than finding the whole plan. The method for determining the lower bound depends on the type of the optimized function.

- Optimization of $f^C_{i,s}$: 0 is the lower bound for the first iteration. It is because in our tests in the most cases there exists an orientation with all color values equal to 0 (see Figure 3). If the first iteration is stopped due to a timeout (i.e. optimality of the solution isn't proved), we create a new problem which is composed only of the configuration i and the side s. Then we find an optimal value of $f^C_{i,s}$ for this small problem. The optimal value from this small problem defines a new lower bound in the full problem (computation of the optimal value of the plan for one configuration is significantly easier than computing optimal values in larger plans).
- Optimization of $f^M_{i,s}$: The lower bound is computed before the first optimization. The method of the computation is similar to the previous case. The only difference is that the fixed value of corresponding $f^C_{i,s}$ should be taken into account during optimization of the small problem, because $f^C_{i,s}$ is optimized prior to $f^M_{i,s}$ and because the constraint on the value of $f^C_{i,s}$ doesn't allow the Autotrack mode unless all tables are monochromatic.
- Optimization of $f^D_{i,s}$ and optimization of $f^L_{i,s}$: 0 is set as the lower bound, because the complete calculation of the lower bound is too time-consuming.

Simplified Models

Solar array planning is a hard problem so we used simplified models which speed up the optimization without significant restriction of the solution space. The simplification of the model is obtained by additional constraints, so every solution in this simplified model is also a solution of the original problem. However, the optimal solution in the simplified model may not be optimal in the original model. So the optimal solution from the simplified model must be validated, i.e., the solution quality serves as the upper bound in the full model. This validation has higher success rate than search from scratch in the full model. We used the following two simplified models.

Our experiments showed that the Autotrack mode has the biggest influence on the difficulty of the problem. It is because this mode causes the orientation at the beginning of the configuration to be different from the orientation at the end and these differences are much harder to predict than differences caused by turnings. Functions $f^C_{i,s}$ are optimized first and they don't take modes into account, so we can omit the Autotrack mode during optimization of $f^C_{i,s}$. It cannot be omitted later, because the Autotrack mode is the best rated mode during optimization of $f^M_{i,s}$ and because functions $f^M_{i,s}$ are optimized prior to $f^D_{i,s}$ and $f^L_{i,s}$. This omission can be done because in most cases the result of the Autotrack mode is the same as the result of the Park mode with turning, and because if turning is prohibited, there is a high chance that the Autotrack mode will be prohibited too.

During turning, each orientation (except the initial orientation) can be achieved in two ways: (a) turning in the positive direction by angle α, and (b) turning in the negative direction by angle $360 - \alpha$. A simplified model can have each orientation accessible in only one way. It can be enforced in two ways:

- The negative direction is prohibited, i.e., $R^x_{i,s,j} \geq 0$. Symmetrically it would be possible to prohibit the positive direction, but the negative direction causes changes of directions in contrast with the positive direction.
- The angle of turnings is restricted to interval $[0; 180]$, i.e., $|R^x_{i,s,j}| \leq 180$. One position is still accessible in two ways – if the angle of turning is $180°$, it doesn't matter whether the direction is positive or negative. However, restriction of the angle to $[0; 180)$ in one direction showed to be less effective.

This simplification is used during optimization of the function $f^M_{i,s}$. We used the second approach, because it gives better results than the first one.

Experiments

Test Cases

There is a set of test cases attached to the definition of the problem (Frank, 2012), but its size and difficulty is insufficient for testing of the algorithm and parameter tuning (our method solved all of them optimally and no turning was necessary in the optimal schedules). Due to confidentiality it was not possible to obtain real data, so we generated a dataset by replacing tables of the most difficult original test case from (Frank, 2012) with newly generated tables. The original test case contains four configurations in total duration seven hours. We created 100 test cases which are more difficult to solve than the original test cases. Tables were generated as follows:

1. Start with a green table.
2. Add a yellow region with probability $\frac{1}{4}$ to the table. The yellow region is a horizontal stripe, a vertical stripe, or a rectangle. Its dimensions are taken randomly from interval $[120; 300]$ with uniform probability.
3. Add a red region with probability $\frac{2}{3}$ to the yellow region. The red region has the same shape as its parental yellow region and yellow margin is at least 30px wide.

The planning horizon is only seven hours long in contrast to the four-week real planning horizon. However, data collected from ISS Live![1] during one month showed that short problematic periods (a few hours) are interspersed with a several-days long periods when all joints are in the mode Autotrack. So in fact, it is necessary to generate plans only for short periods, because joints are in the mode Autotrack in the rest of the time. Figure 7 shows one case where some joints aren't in the mode Autotrack.

[1] http://spacestationlive.nasa.gov/

Parameter Tuning

Our algorithm contains two parameters that need to be tuned. The first one is the labeling strategy used in the search algorithm and the second one is the value of the time out.

We use SICStus Prolog and its CLPFD library (Carlsson *et al.*, 1997) for implementation of the solar array planing model, so we exploited its variable ordering heuristics (leftmost, ff, ffc, min, max) and branching schemes (step, bisect) too. It might be appropriate to develop heuristics specialized for the problem of solar array planning, but general heuristics were able to optimize all original test cases. We do not use all combinations of variable ordering heuristics and branching schemes, because that is time consuming; we selected only a few pairs with the best results in initial experiments. We use two pairs during optimization of $f^C_{i,s}$ (namely (max,step) and (ff,bisect)) and $f^M_{i,s}$ (namely (ffc,bisect) and (leftmost,bisect)), one pair (min,bisect) during optimization of $f^D_{i,s}$, and two pairs (leftmost,bisect) and (min,bisect) during optimization of $f^L_{i,s}$.

As time outs were chosen 80ms for optimizing each of $f^C_{i,s}$ and $f^M_{i,s}$, 160ms for optimization of $f^D_{i,s}$, and 800ms for optimization of $f^L_{i,s}$. These values were chosen based on the experiments. The timeout is applied to every abovementioned combination of heuristics.

Results

We compare the proposed algorithm "Final" with some alternative approaches. The results are summarized in Table 1. All optimization algorithms were implemented in C# with .NET Framework 4 and they use the CLPFD library from SICStus Prolog for labeling of variables. All algorithms except of "Greedy" use the same time outs (see above), "Greedy" does not use time outs. Testing was performed on the laptop with an Intel® Core™ i5-520M processor (2.40GHz) and 8GB PC3-8500 DDR3 SDRAM.

First we compare the "Final" algorithm with a variation "Wider" that uses a full portfolio of variable ordering heuristics and branching schemes provided by the CLPFD library (five variable ordering heuristics and three branching schemes) to show that we chose an appropriate subset of available heuristics. The "Final" algorithm is only 1% worse in the score of $f^M_{i,s}$, so there is only little space for improvement. The difference in $f^L_{i,s}$ is more significant; the "Wider" is about 12% better, but the "Wider" is still not able to optimize 14 plans, i.e., it finds only five more optimal plans. Moreover, each expansion of the portfolio of heuristics of the "Final" will have a negative effect on the runtime.

"Perpendicular" is the algorithm that optimizes plans in configurations, i.e., all objective functions of one configuration are optimized before any objective function of another configuration is optimized – the order of objective functions is $(f^C_{0,s}, f^M_{0,s}, f^D_{0,s}, f^L_{0,s}, f^C_{1,s}, f^M_{1,s}, \ldots, f^D_{n-1,s}, f^L_{n-1,s})$ – so it is more similar to left-to-right scheduling of the SACE approach, but it isn't a greedy algorithm. It has about 41% better score than "Final" in the least important part of the objective function at the expense of the most important part of the objective function, where it has about 111% worse score. So these plans put more strain on solar arrays.

Method	*Time [s]*	*Number of time outs*				*Average value in final plans*			
		$f^C_{i,s}$	$f^M_{i,s}$	$f^D_{i,s}$	$f^L_{i,s}$	$f^C_{i,s}$	$f^M_{i,s}$	$f^D_{i,s}$	$f^L_{i,s}$
Final	1075	0	10	7	19	1.02	3.24	0.01	36.61
Wider	19758	0	2	8	14	1.02	3.19	0.01	32.19
Perpendicular	1369	8	5	1	7	2.15	3.21	0.01	21.57
Greedy	725	0	0	0	0	1.71	3.68	0.00	87.24

Table 1: Comparison of different approaches. Time is the total time in seconds to solve all 100 scenarios. Number of time outs is the number of plans out of 100 that a given approach does not consider as optimal because of insufficient time for the optimization. The greedy algorithm can claim a suboptimal solution as the optimal solution, because this algorithm can cut the branch with the optimal solution off at some stage of search. The average value in final plans shows the average value of the corresponding part of the objective function across all scenarios (all functions are minimized). Scores of colors $f^C_{i,s}$ are from domain [0; 92], scores of modes $f^M_{i,s}$ are from domain [0; 10], numbers of changes of direction $f^D_{i,s}$ are from domain [0; 15] and duration of turning $f^L_{i,s}$ are from domain [0; 3590]. "Final" is the algorithm proposed in this paper. "Wider" is a variation of "Final" with wider portfolio of variable ordering heuristics and branching schemes. "Perpendicular" is the algorithm that optimizes objectives in the order $f_{0,s}, f_{1,s}, \ldots, f_{n-1,s}$. "Greedy" is our implementation of the SACE approach.

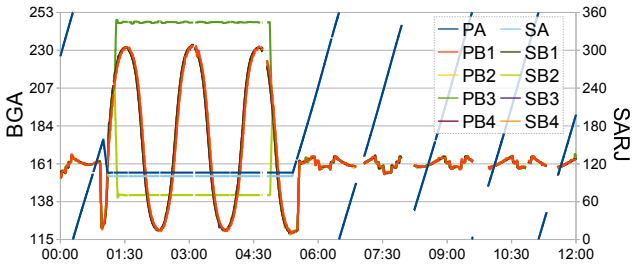

Figure 7: Example of real orientations of joints during half a day (data collected from ISS Life!).

"Greedy" is our implementation of the SACE approach based on (Reddy *et al.*, 2011), because their implementation isn't publicly accessible. The algorithm goes greedily from left to right and searches the optimal orientation for the current configuration according to fixed orientations of all previous configurations. If the algorithm fails in an attempt to extend the schedule, it merges the current configuration with the previous one. Because the algorithm searches the space of a constant size during the optimization of a single configuration, the computational time required for the optimization of a single configuration is upper bounded and no time outs are needed. It needs only about 33% less time for optimization of all plans, but the average score is significantly worse in all parts of the objective function except $f^D_{i,s}$. The average score of $f^C_{i,s}$ is about 67% worse, the average score of $f^M_{i,s}$ is about 14% worse and the average score of $f^L_{i,s}$ is about 138% worse. So our approach is able to provide significantly better plans in a slightly longer time.

Concluding Remarks

This paper presents a constraint-based optimizer for scheduling solar array operations (COSSA) at the International Space Station. In particular, we proposed a constraint model with highly complex transition constraints and objectives and a special layered optimization algorithm exploiting a portfolio of classical variable ordering heuristics and branching schemes. We experimentally showed that this approach generates much better schedules than the classical left-to-right scheduling while keeping similar time efficiency. Though this global optimization approach does not scale yet to a full four-week horizon, based on data collected from ISS, the periods, when something happens and scheduling is complex, are short enough to be covered by our optimizer. The major contribution is showing that off-the-shelf technology with some specialized optimization procedure overcomes a specialized solver in terms of schedule quality while keeping time efficiency comparable. Full technical details are available in (Jelínek, 2014).

Acknowledgement

Roman Barták is supported by the Czech Science Foundation under the project P103-15-19877S.

References

Jelínek, J., 2014. Plánování operací solárních panelů na ISS (in Czech). Diploma Thesis, Charles University in Prague.

Carlsson, M.; Ottosson, G.; and Carlsson, B. 1997. An Open-Ended Finite Domain Constraint Solver. In *Programming Languages: Implementations, Logics, and Programs*.

Frank, J. D. 2012. Planning Solar Array Operations on the International Space Station. In *The International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012)*.

Frank, J., and Jónsson, A. 2003. Constraint-Based attribute and interval planning. *J. Constraints* 8(4): 339–364.

Land, A. H., and Doig, A. G. 1960. An automatic method of solving discrete programming problems. In *Econometrica* 28(3), 497–520.

Reddy, S.; Frank, J.; Iatauro, M.; Boyce, M.; Kürklü, E.; Ai-Chang, M.; Jónsson, A. 2011. Planning Solar Array Operations for the International Space Station. In *ACM Transactions on Intelligent Systems and Technology* 2(4).