ICAPS 2006

The English Lake District, Cumbria, UK

VIS6

Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

Miguel A. Salido
Antonio Garrido
Universidad Politécnica de Valencia, Spain

Alexander Nareyek
Digital Drama Studios, Czech Republic

Maria Fox
University of Strathclyde, UK














ICAPS 2006

The English Lake District, Cumbria, UK

Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

WS6

Miguel A. Salido
Antonio Garrido
Universidad Politécnica de Valencia, Spain

Alexander Nareyek
Digital Drama Studios, Czech Republic

Maria Fox
University of Strathclyde, UK

ICAPS 2006
Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

Table of contents

ICAPS 2006
Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

Preface

The areas of AI planning and scheduling have seen important advances thanks to the constraint satisfaction techniques. Now, many important real-world problems require dealing with efficient constraint techniques for planning, scheduling and resource allocation to competing goal activities over time in the presence of complex state-dependent constraints. Therefore, solutions to these problems must integrate resource allocation and plan synthesis capabilities. Basically, we need to manage complex problems where planning, scheduling and constraint satisfaction must be interrelated, which entail a great potential of application.

The workshop therefore aims at providing a platform for meeting and exchanging ideas and novel works in the field of AI planning, scheduling, constraint satisfaction techniques, and many other common areas that exist among them. In fact, most of the received works are based on combined approaches of constraint satisfaction for planning, scheduling and mixing planning and scheduling. The workshop will be held in June, 2006 in Cumbria (United Kingdom) as one of the six workshops at the International Conference on Automated Planning & Scheduling (ICAPS'06).

This book contains the proceedings of the workshop which comprise both theoretic and applied papers. The number of submitted papers was 15. All the papers were reviewed by two anonymous referees from the program committee, who decided to accept 7 as long papers and 2 as short papers, all for oral presentation in the workshop. The papers provide a good mix of constraint satisfaction techniques for planning, scheduling, related topics and their applications to real-world problems. We hope that the ideas and approaches presented in the papers and presentations will lead to a valuable discussion and will inspire future research and developments in all the workshop participants.

The Organizing Committee April, 2006

Organizers

- *Miguel A. Salido, Universidad Politcnica de Valencia (Spain)*
- *Antonio Garrido, Universidad Politcnica de Valencia (Spain)*
- *Alexander Nareyek, Digital Drama Studios (Czech Republic)*
- *Maria Fox (additional collaboration), University of Strathclyde (United Kingdom)*

Programme Committee

- *Federico Barber Universidad Politcnica de Valencia (Spain)*
- *Roman Bartak Charles University (Czech Republic)*
- *Chris Beck University of Toronto (Canada)*
- *Luis Castillo Universidad de Granada (Spain)*
- *Amedeo Cesta ISTC-CNR (Italy)*
- *Maria Fox University of Strathclyde (UK)*
- *Hector Geffner Universidad Pompeu Fabra (Spain)*
- *Enrico Giunchiglia Universit di Genova (Italy)*
- *Derek Long University of Strathclyde (UK)*
- *Pedro Meseguer IIIA-CSIC (Spain)*
- *Alexander Nareyek Digital Drama Studios (Czech Republic)*
- *Eva Onaindía Universidad Politcnica de Valencia (Spain)*
- *Pascal Van Hentenryck Brown University (USA)*
- *Vincent Vidal CRIL-IUT (France)*
- *Toby Walsh UNSW, Sydney and NICTA (Australia)*

Additional Reviewers

- *Antonio Garrido Universidad Politcnica de Valencia (Spain)*
- *Emmanuel Hebrard UNSW, Sydney and NICTA (Australia)*
- *Federico Pecora ISTC-CNR (Italy)*
- *Miguel A. Salido Universidad Politcnica de Valencia (Spain)*
- *Oscar Sapena Universidad Politcnica de Valencia (Spain)*

Factored Planning: How, When, and When Not

Ronen I. Brafman*
Department of Computer Science
Stanford University, CA, USA
brafman@cs.stanford.edu

Carmel Domshlak
Faculty of Industrial Engineering and Management
Technion, Israel
dcarmel@ie.technion.ac.il

Abstract

Automated domain factoring, and planning methods that utilize them, have long been of interest to planning researchers. Recent work in this area yielded new theoretical insight and algorithms, but left many questions open: How to decompose a domain into factors? How to work with these factors? And whether and when decomposition-based methods are useful? This paper provides theoretical analysis that answers many of these questions: it proposes a novel approach to factored planning; proves its theoretical superiority over previous methods; provides insight into how to factor domains; and uses its novel complexity results to analyze when factored planning is likely to perform well, and when not. It also establishes the key role played by the domain's causal graph in the complexity analysis of planning algorithms.

Introduction

Factored planning is a collective name for planning algorithms that exploit independence within a planning problem to *decompose* the domain, and then work on each subdomain (= factor) separately while trying to piece the constructed sub-plans into a valid global plan. Hierarchical planners (Knoblock 1994; Lansky & Getoor 1995) are probably the best-known examples of such algorithms. They vertically factor the domain into a set of increasingly more detailed abstraction levels. They plan in each level separately while reusing the solution of more abstract levels. The problem, however, is that hierarchical decomposition works well only in domains where one component's value has little direct *and* indirect influence on that of others. When such structure is missing, abstraction-generation techniques such as (Sacerdoti 1974; Knoblock 1994) yield no or only minor decomposition, and backtracking between sub-domains in the latter case can dominate the complexity of solving the non-decomposed problem.

The spectacular improvement in standard planning algorithm over the past decade, together with the above limitations of vertical factoring, pushed factored planning to the backstage of domain-independent planning research.[1] Recently, however, this situation has somewhat changed. First, a number of papers on the formal complexity of planning as a function of certain factored decomposition appeared (Brafman & Domshlak 2003; Domshlak & Dinitz 2001). Second, recent developments in heuristic-search planning have shown that factored problem decompositions and abstractions can provide extremely effective heuristic guidance (Helmert 2004). Finally, recent work by Amir & Engelhardt 2003 (henceforth referred to as AE) has produced a systematic, general-purpose approach to factored planning, with a clear worst-case complexity analysis.

This recent evolution of work on domain-independent factored planning leaves open two major questions. The first question is *how*, i.e., what is the best way to decompose a problem? Previous factoring methods used various graphical structures to drive the factorization process. The structure of such a graph is a significant parameter in the success of each method. Hence, finding a graphical structure leading to a provably better (or even optimal) factorization is clearly of interest. The second, closely related question is *when*: When should factored planning be expected to work better than standard planning. Addressing this question requires better understanding of the complexity of factored and non-factored planning and the parameters affecting them.

In this paper we address these two questions of *how* and *when* through the lens of worst-case complexity analysis. We identify the domain's *causal graph* as an essential structure in the analysis of factored planning, showing that it captures all the sufficient and necessary information about variable interactions. In particular, we show that our approach based on causal graphs is strictly more efficient (by up to an exponential factor) than the AE approach. We show that the tree-width of causal graphs plays a key role in the complexity of both our approach to factored planning, as well as existing methods for *non-factored* step-optimal planning. This finding allows us to relate factored and non-factored methods and understand when each is likely to work best.

Background

We start with a few basic definitions of the planning problem as defined in the SAS$^+$ formalism (Bäckström & Nebel

*On leave from Ben-Gurion University.

[1] Here we refer to methods that automatically induce a hierarchy from domain description, unlike HTN planning (Erol, Hendler, & Nao 1994) where the hierarchy provides additional domain knowledge that often significantly improves performance.

1995) followed by the definition of the causal graph. The SAS$^+$ formalism models domains using multi-valued state variables. It distinguishes between *pre*-conditions and *prevail* conditions of an action. The former are required values of variables that are affected by the action. The latter are required values of variables that are not affected by the action. The post-conditions of an action describe the new values its precondition variables. For example, having a visa is a prevail condition for applying the action Enter-USA, while having a valid ticket is a precondition of the action Fly-To-USA, as its value changes from true to false following the action's execution. An action is applicable if and only if both its pre- and prevail conditions are satisfied.

Definition 1 *A* SAS$^+$ *problem instance is given by a quadruple* $\Pi = \langle V, A, I, G \rangle$*, where:*

- $V = \{v_1, \ldots, v_n\}$ *is a set of* state variables *with finite* domains $dom(v_i)$*. The domain* $dom(v_i)$ *of the variable* v_i *induces an* extended domain $dom^+(v_i) = dom(v_i) \cup \{\mathsf{u}\}$*, where* u *denotes the value:* unspecified.
- I *is a fully specified* initial state*, that is,* $I \in \times dom(v_i)$*. By* $I[i]$ *we denote the value of* v_i *in* I*.*
- G *specifies a set of alternative* goal states*. Adopting the standard practice in the planning research, we assume that such a set is specified by a partial assignment on* V*, that is,* $G \in \times dom^+(v_i)$*. By* $G[i]$ *we denote the value provided by* G *to* v_i *(with, possibly,* $G[i] = \mathsf{u}$*.)*
- $A = \{a_1, \ldots, a_m\}$ *is a finite set of* actions*. Each action* a_i *is a tuple* $\langle \mathsf{pre}(a_i), \mathsf{post}(a_i), \mathsf{prv}(a_i) \rangle$*, where* $\mathsf{pre}(a_i), \mathsf{post}(a_i), \mathsf{prv}(a_i) \subseteq \times dom^+(v_i)$ *denote the* pre-, post-*, and* prevail conditions *of* a*, respectively. In what follows, by* $\mathsf{pre}(a)[i]$*,* $\mathsf{post}(a)[i]$*, and* $\mathsf{prv}(a)[i]$ *we denote the corresponding values of* v_i*.*

The factorization of planning problems we propose here is based on the well-known *causal graph* structure (Bacchus & Yang 1994; Knoblock 1994; Brafman & Domshlak 2003; Domshlak & Dinitz 2001; Helmert 2004; Williams & Nayak 1997).

Definition 2 *Given a planning problem* $\Pi = \langle V, A, I, G \rangle$*, the* causal graph *CG*$_\Pi$ *of* Π *is a mixed (directed/undirected) graph over the nodes* V*. A directed edge* $(\overrightarrow{v_i, v_j})$ *appears in CG*$_\Pi$ *if (and only if) some action in* A *that changes the value of* v_j *has a prevail condition involving some value of* v_i*. An undirected edge* (v_i, v_j) *appears in CG*$_\Pi$ *if (and only if) some action in* A *changes the values of* v_i *and* v_j *simultaneously.*

Informally, the immediate predecessors of v in *CG*$_\Pi$ are all those variables that directly affect our ability to change the value of v. It is worth noting that nothing in Definition 2 prevents us from having for some pair of variables $v_i, v_j \in V$ in *CG*$_\Pi$ both $(\overrightarrow{v_i, v_j})$, and $(\overrightarrow{v_j, v_i})$, and (v_i, v_j). In any case, it is evident that constructing the causal graph *CG*$_\Pi$ of any given SAS$^+$ planning problem Π is straightforward.

Example 1 *Suppose we have two packages,* A *and* B*, a rocket, and two locations,* E *and* M*. Packages can be either in a location or in the rocket, and the rocket requires fuel to fly. The actions correspond to loading and unloading the packages, flying the rocket, and fueling the rocket. Flying the rocket consumes the fuel, but it can be fueled in any location. We model this problem in the* SAS$^+$ *formalism as follows. The variables are* r, a, b, f*;* r *denotes the position of the rocket and its domain is* at-E*,* at-M*.* a *and* b *denote positions of the packages* A *and* B *and their domains are:* at-E, at-M, at-rocket*.* f *denotes whether or not the rocket has fuel with values* full *and* empty*. For* $x \neq y \in \{$e,m$\}$*, the actions are* fly-x-y, load-x-y, unload-x-y*, and* fuel*. The* fly-x-y *actions have two precondition –* f=full *and* r=at-x*. Their post-conditions are* f=empty *and* r=at-y*. The* load-x-y *actions have one precondition –* x=at-y *– and one prevail condition –* r=at-y*. The single post-condition is* x=at-rocket*. Finally, the action* fuel *has one pre-condition and one post-condition:* empty *and* full *respectively. The causal graph for this problem is shown in Figure 1.*

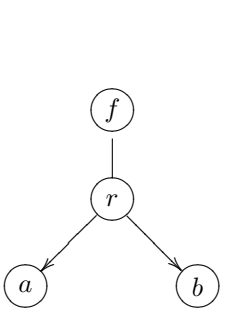


Figure 1: Causal graph for Example 1.

Sequence-Based CSP Planning

The central questions for any factored planning approach are how to decompose the problem and how to piece together solutions from different sub-domains. Our initial answer to the first question, which later we generalize, is very simple: factor = variable. We can now focus on the questions of how to combine a set of given local plans for different factors, and then, how to generate these local plans. The causal graph plays a key role in the algorithm we propose. Its tree-width plays an equally important role in the complexity analysis of the algorithm.

Locally-Optimal Factored Planning

Let $A_i \subseteq A$ denote the set of all actions *affecting* $v_i \in V$. Suppose that, for every v_i, we are given a set of *pre-scheduled action sequences* $SPlan(v_i)$ where each $\rho_i \in SPlan(v_i)$ is a finite sequence of pairs (a, t) with $a \in A_i$, and $t \in \mathbb{Z}^+$ is the time point at which a is to be performed. We now ask ourselves how we might construct plans using these n sets of action sequences $SPlan(v_i)$. A key observation is that this particular problem can be solved by compiling it into a binary CSP (denoted *SeqCSP*) over n variables $X_1, \ldots, X_n$ where:

1. The domain of X_i is exactly $SPlan(v_i)$, and
2. The constraints of *SeqCSP bijectively* correspond to the edges of the causal graph *CG*$_\Pi$.

Informally, the constraint corresponding to a directed edge $(\overrightarrow{v_i, v_j}) \in$ *CG*$_\Pi$ ensures that the action sequence selected for v_i provides all the prevail conditions required by the actions of the sequence selected for v_j, and that the timing of

```
procedure LID
  d := 1
  loop
    for i := 1 ... n do
        Dom(X_i) := all sub-plans for v_i of length up to d,
                    over all schedules across nd time points.
    Construct SeqCSP_Π(d) over X_1, ..., X_n.
    if ( solve-csp(SeqCSP_Π(d)) ) then
        Reconstruct a plan ρ from the solution of SeqCSP_Π(d).
        return ρ
    else
        d := d + 1
  endloop
```

Figure 2: Factored planning via local iterative deepening.

this provision is correct. The constraint corresponding to an undirected edge $(v_i, v_j) \in CG_\Pi$ ensures that the restrictions of the sequences selected for v_i and v_j to $A_i \cap A_j$ are identical. This causal-graph based problem reformulation allows us to formalize the worst-case complexity of solving our "sequences combination" problem in a structure-informed manner. Since each constraint of *SeqCSP* can be verified in polynomial time, from a classical result on tractable CSPs (Dechter 2003) we have that the time complexity of the "sub-plans combination" problem is $O(n\sigma^{w+1})$, where w is the tree-width of the undirected graph underlying CG_Π, and $\sigma = \max_i \{|SPlan(v_i)|\}$.

While in itself our "sequences combination" problem is not of general interest, the distance between its CSP formulation, and that of general planning problems is not large. Specifically, given a planning problem $\Pi = \langle V, A, I, G\rangle$, one can solve it using the LID (short for *local iterative deepening*) procedure depicted in Figure 2. LID searches for a plan by performing local iterative deepening on the maximal number of changes that a plan might induce on a single state variable. Given such an upper bound $d \geq 1$, LID formulates a constraint satisfaction problem $SeqCSP_\Pi(d)$ where, for $1 \leq i \leq n$, $SPlan(v_i)$ contains all (consistent with I and G) sequences of length at most d of actions affecting v_i. Each such sequence is considered with respect to all possible time schedules of its actions. Since each state variable in iteration d is allowed to change its value up to d times, it is sufficient to consider a time horizon of nd. If, at some iteration d, $SeqCSP_\Pi(d)$ is solvable, a valid plan (containing no idle time points) can be easily extracted from the corresponding solution of $SeqCSP_\Pi(d)$.

Theorem 1 LID *is a sound and complete planning algorithm. Moreover, if* LID *terminates with a plan ρ at iteration d, then, for any other plan ρ' for the considered problem instance, there exists a state variable that changes its value on ρ' at least d times.*

The CSP encoding used in LID may seem a bit crude, but it is simple to understand, and all the essential ideas and formal results of this work already fall out from it. Later, however, we describe an *equivalent*, yet technically more involved, encoding in the spirit of standard planning-as-CSP encodings. For the next result we introduce the following notation: Let $Plan(\Pi)$ be the (possibly infinite) set of all plans for Π. For each plan $\rho \in Plan(\Pi)$, and each $1 \leq i \leq n$, let ρ_i denote the subset of all actions in ρ affecting variable v_i. Finally, let G_{CG} denote the undirected graph underlying causal graph CG_Π.

Theorem 2 *Given a planning problem Π, it can be solved using* LID *in time*

$$O(n(n\delta a)^{w\delta+\delta}) \tag{1}$$

where $a = \max_i \{|A_i|\}$, w is the tree-width of G_{CG}, and δ is the local depth *of Π defined as:*

$$\delta = \min_{\rho \in Plan(\Pi)} \max_{1 \leq i \leq n} \{|\rho_i|\} \tag{2}$$

Theorem 2 expresses[2] the complexity of LID in terms of two parameters. The tree width of the domain's causal graph measures the level of interaction between the domain variables. The parameter δ is problem-instance dependent and it expresses the minmax amount of work required on a single variable. In particular, we note that Theorem 2 establishes a new tractable class of planning problems, because for problems with both w and δ bounded by some constants, Eq. 1 trivially reduces to a polynomial.

Generalized Factoring

So far, we assumed that factor = variable, yet it is not clear that this factorization leads to the best possible worst-case performance. Here we take a closer look at this question by drawing on our previous analysis to understand possible effects of using a different factoring.

Two parameters affect the worst-case complexity of factored planning: tree-width and minmax number of changes per factor (local depth, for factor=variable.) Thus, we need to understand the effect of alternative factorizations on these parameters. Consider variables $v_1, \ldots, v_k$ that change their value $c_1, \ldots, c_k$ times in a locally optimal plan when single variable factors are used. If we combine these variables into a single factor, this new factor will change its value at most $\sum_{i=1}^{k} c_i$ times in any locally optimal (for the new factorization) plan. In general, it is not hard to verify that the minmax number of changes per factor under factorization with maximal factor size k could be as large as $k\delta$.

While this seems like a big loss, observe that it can be offset by a reduction in the tree-width of the constraint graph. Indeed, it is well known that for each CSP whose primal graph has tree-width w, there is a tree-decomposition with maximal node size $w + 1$. Such a tree-decomposition defines an equivalent CSP whose variables (our new factors) are cross-products of the original variables, and whose constraint graph forms a tree[3], that is, has tree-width of 1. We also already know that d_f, the minmax number of value changes per new factor, is upper bounded by $(w+1)\delta$. However, observe that it can also be much better. For any tree-decomposition, we know that d_f is bounded by the maximal

[2]For the proofs of this and other formal claims in the paper, we refer the reader to the full version.

[3]Though constructing an optimal tree-decomposition (i.e., one with maximal node size $w + 1$) is NP-hard (Arnborg, Cornell, & Proskurowski 1987), there are numerous effective, fast approximation and heuristic algorithms for this problem.

sum of value changes of original variables in any new factor. If so, then unless all the variables clustered together have to change δ times each, d_f would be less than $(w+1)\delta$, and possibly much less, down to δ!

Consequently, we can adapt LID to any tree-decomposition, and any form of factoring. Instead of iterating over the maximal number of value changes of a variable, we iterate over the maximal number of value changes of a factor, i.e., a node of the given tree-decomposition. We refer to this procedure as LID-GF, and its complexity is described by Theorem 3.

Theorem 3 *Given a planning problem Π, the time complexity of solving it using* LID-GF *on an optimal tree-decomposition of G_{CG} is $O\left(n(wa+a)^{d_f}\right)$, where w is the treewidth of G_{CG}, $a = \max_i \{|A_i|\}$, and $d_f \leq (w+1)\delta$.*

It is now apparent that by moving from the extreme of factor = variable to an optimal tree-decomposition, we cannot lose, and are most likely to improve our worst-case complexity. The complexity now changes from exponential in $w\delta$ to something that is at least as good as wd_{avg}, where d_{avg} is the maximal (over the factors) average number of variable changes within the factor. Thus, when constructing a tree-decomposition, one needs to consider both the cluster size and its variability, where the value to keep in mind is the (unknown) sum of value changes of variables in a cluster. The good news is that even if we know nothing about the domain, Theorems 2 and 3 imply that we cannot lose by moving to an optimal tree-decomposition.

Perhaps the better news is that we have here a concrete role for domain knowledge. Suppose we have some idea about which variables are likely to change a lot and which variables are likely to change just a little. In that case, we can impose some constraints on the tree-decomposition, ensuring that certain variables appear together in it. We can do this by constructing a *constrained* tree-decomposition, that is, a tree decomposition in which we *a priori* require certain problem variables to be together. This could lead us to tree decomposition with larger nodes, but with smaller sum-of-value-changes, leading to improved performance.

Comparison with AE

Having our generalized LID-GF approach, we now show that it provides better complexity guarantees than AE, a recently proposed approach to factored planning with the first clear complexity analysis (Amir & Engelhardt 2003).

Similarly to LID-GF, AE has a single factoring phase, followed by a sequence of planning phases invoked in an iterative deepening fashion over the upper-bound on the depth of the local plans. The factoring phase takes a certain graph induced by the given problem instance, and constructs a tree decomposition of this graph (named here AEG_Π) using one of the off-the-shelf algorithms for close-to-optimal tree decomposition. Given such a tree of planning factors (each factor corresponding to a subset of state variables), each planning phase processes this tree incrementally in a bottom-up fashion. In processing each sub-domain, AE looks for a local plan of a bounded depth over a certain set of complex macro actions. The search for local plans is performed using a generic black-box planner.[4]

Though algorithmically different, both LID-GF and AE use local iterative deepening to search for plans, and provide *similar guarantees on the quality* of the resulting plan. That is, plans returned by both approaches are guaranteed to be locally optimal at the level of factors of tree decomposition in use. However, the worst-case complexity of these two approaches is not the same. First, while both approaches scale linearly in the number of state variables, the worst-case complexity of AE grows exponentially in $w_{ae}d_f = \Theta(w_{ae}^2\delta)$ where w_{ae} is the tree-width of AEG_Π, while that of LID-GF grows exponentially in $wd_f = \Theta(w\delta)$. Assuming for a moment that the tree-width of the causal graph and this of AEG_Π are comparable, this already shows that LID-GF is worst-case more efficient than AE. However, Theorem 4 shows that the actual difference is much larger, and that it can be exponential in $\Theta(n)$.

Theorem 4 *Given a planning problem Π, let w be the tree-width of G_{CG}, and w_{ae} be the tree-width of AEG_Π. For all planning problems Π, we have $w_d \leq w_{ae}$, and there are problems for which we have $w_d = O(1)$ and $w_{ae} = \Theta(n)$.*

Such a gap between the time complexity of LID-GF and AE stems from the structure of the dependencies between the state variables that these two approaches exploit. While problem decomposition in LID-GF is based on the causal graph, AEG_Π is an undirected graph over the nodes V, containing an edge (v_i, v_j) iff there is an action $a \in A$ that *somehow* involves both v_i and v_j, that is,

$$(\mathsf{pre}(a)[i] \neq \mathsf{u} \vee \mathsf{post}(a)[i] \neq \mathsf{u} \vee \mathsf{prv}(a)[i] \neq \mathsf{u}) \wedge$$
$$(\mathsf{pre}(a)[j] \neq \mathsf{u} \vee \mathsf{post}(a)[j] \neq \mathsf{u} \vee \mathsf{prv}(a)[j] \neq \mathsf{u})$$

Given that, it is easy to verify that G_{CG} is a *subgraph* of AEG_Π, and thus $w \leq w_{ae}$.

To show the potentially linear difference between w and w_{ae}, consider the following problem Π' over (possibly propositional) variables $v_1, \ldots, v_n$:

- $v_1, \ldots, v_{n-1}$ can each be changed independently (and only independently) of the rest of the variables, i.e.,

$$\forall 1 \leq i \leq n-1, \forall a \in A_i, \forall 1 \leq j \leq n.\ \mathsf{prv}(a)[j] = \mathsf{u}$$

- For each pair of variables from $v_1, \ldots, v_{n-1}$, there exists an action changing the value of v_n prevailed by an assignment to this pair of variables, that is:

$$\forall 1 \leq i \neq j \leq n-1\ \exists a \in A_n.\ \mathsf{prv}(a)[i] \neq \mathsf{u} \wedge \mathsf{prv}(a)[j] \neq \mathsf{u}$$

Here, G_{CG} forms a tree (that is, $w = 1$), while AEG_Π forms a clique of all n nodes, and thus $w_{ae} = n$.

Factoring and Plan Optimality

Classical planning offers a few notions of plan optimality, with the most standard being sequential optimality (henceforth, OP), which corresponds to a plan with a minimal number of actions. Step-optimal planning (SOP) is an alternative that stands for minimizing the number of time steps in which

[4] For detailed description of AE, see (Amir & Engelhardt 2003).

a plan can be executed under a valid parallelizing of its actions. Depending on the application, SOP can be either of interest on its own, or considered as a reasonable compromise when OP is beyond reach. We argue that, from this perspective, the notion of local optimality (LOP) targeted by factored planners is not any different. In some applications, LOP is of interest on its own, e.g., in the context of distributed systems. And viewed as an approximation to OP, LOP and SOP provide similar guarantees, as shown below.

Lemma 1 *Given a planning problem Π, let m_{op}, m_{sop}, m_{lop} denote the number of actions in an optimal, step-optimal, and locally optimal plan, respectively. We have that $m_{sop} \leq n \cdot m_{op}$ and $m_{lop} \leq n \cdot m_{op}$ (where n is the number of variables in Π), and both these bounds are tight.*

Given the "approximation equivalence" between SOP and LOP established by Lemma 1, we turn to consider the time complexity guarantees of standard methods for OP, SOP, and LOP. To the best of our knowledge, such worst-case time guarantees for OP are either exponential in the length of the optimal plan (e.g., state-space forward search using BFS), or exponential in the problem size (e.g., planning-as-CSP with a linear encodings (Kautz & Selman 1996)). At this point, for SOP, all methods with established complexity guarantees are of the second type, that is, worst-case exponential in the problem size – we will have something to say about this later. Thus, moving from OP to SOP appears to buy us nothing in terms of *formal* bounds on the time complexity. The situation with LOP, however, is different. Theorem 2 shows that the direct dependence of LID's complexity on *both* the problem size and plan length is polynomial. The exponential dependence of LID is on two other, deeper problem characteristics, namely the tightness of problem structure (w), and the amount of local effort required on each problem factor in order to solve the problem (δ).

Below we take a closer look at the relationship between the complexity guarantees for LOP, SOP, and OP. In the course of this comparative analysis, we provide and exploit some new results on the complexity of SOP. In particular, these results show that in certain situations SOP can actually provide better upper bounds on time complexity than OP. Moreover, these results emphasize the importance of the causal graph in the analysis of planning, as its tree-width plays an important role in the analysis of SOP, as well.

Complexity of SOP using DK

To make our discussion concrete, we consider a characteristic planning-as-CSP approach to SOP described in (Do & Kambhampati 2001) (named here DK). While describing the DK encoding, we ignore the use of graphplan in DK to obtain reachability information in form of temporal mutexes. We make this simplification to separate between the core of the methods and their various possible extensions. The DK encoding is parameterized by an upper bound, m, on the step-length of a plan. Given m, the DK encoding includes a single variable $v^{[k]}$ for every problem variable v and every time stamp $1 \leq k \leq m$. The domain of each variable $v^{[k]}$ is the set of actions that can change the value of v. For any $1 \leq k \leq m$, the value of all variables $v^{[k]}$ encode the state of the system at time k. The following (binary) constraints are imposed:

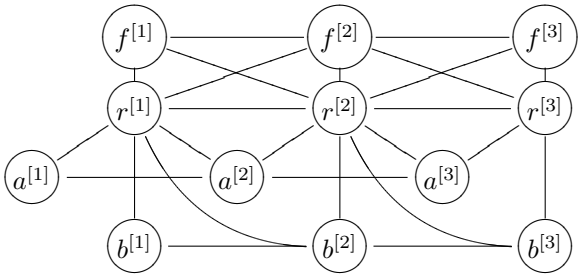


Figure 3: Primal graph $G^{(d)}_{DK}$ of *DK-CSP*$_\Pi(3)$ for example.

- *Initial state*: If $v_i^{[1]} = a$, then $\mathsf{pre}(a) \cup \mathsf{prv}(a) \subseteq I$.
- *Goals:* If $v_i^{[m]} = a$ and $G[i] \neq \mathsf{u}$, then $\mathsf{post}(a)[i] = G[i]$.
- *Precondition:* If $v_i^{[k]} = a$ and $v_i^{[k-1]} = a'$, then $\mathsf{post}(a')[i] = \mathsf{pre}(a)[i]$.
- *Prevail condition:* If $v_i^{[k]} = a$, $v_j^{[k-1]} = a'$, $v_j^{[k]} = a''$, and $\mathsf{prv}(a)[j] \neq \mathsf{u}$, then $\mathsf{post}(a')[j] = \mathsf{post}(a'')[j] = \mathsf{prv}(a)[j]$.
- *Simultaneity:* If $a \in A_i \cap A_j$, then $v_i^{[k]} = a$ iff $v_j^{[k]} = a$.

Example 2 *Consider the Rocket domain once again, and let $m = 3$. The* DK *encoding of this domain is as follows.*

- Initial-state*: Set $dom(f^{[1]}) = \{\mathsf{noop}_{full}, fly\text{-}e\text{-}m\}$, $dom(r^{[1]}) = \{\mathsf{noop}_{r=e}, fly\text{-}e\text{-}m\}$, etc.*
- Goal-state*:*
  For A, we set $dom(a^{[3]}) = \{\mathsf{noop}_{a=m}, unload\text{-}a\text{-}m\}$.
- Precondition *relevant to $r^{[2]}$:*

$$\begin{aligned} r^{[2]} \in \{\mathsf{noop}_{r=e}, fly\text{-}e\text{-}m\} &\rightarrow r^{[1]} \in \{\mathsf{noop}_{r=e}, fly\text{-}m\text{-}e\} \\ r^{[2]} \in \{\mathsf{noop}_{r=m}, fly\text{-}m\text{-}e\} &\rightarrow r^{[1]} \in \{\mathsf{noop}_{r=m}, fly\text{-}e\text{-}m\} \end{aligned}$$

- Prevail *relevant to $a^{[2]}$:*

$$\begin{aligned} a^{[2]} \in \{load\text{-}e, unload\text{-}e\} &\rightarrow r^{[1]} \in \{\mathsf{noop}_{r=e}, fly\text{-}m\text{-}e\} \wedge \\ &\quad\ r^{[2]} \in \{\mathsf{noop}_{r=e}\} \\ a^{[2]} \in \{load\text{-}m, unload\text{-}m\} &\rightarrow r^{[1]} \in \{\mathsf{noop}_{r=m}, fly\text{-}e\text{-}m\} \wedge \\ &\quad\ r^{[2]} \in \{\mathsf{noop}_{r=m}\} \end{aligned}$$

- Simultaneity*:*

$$\begin{aligned} r^{[2]} = fly\text{-}e\text{-}m &\leftrightarrow f^{[2]} = fly\text{-}e\text{-}m \\ r^{[2]} = fly\text{-}m\text{-}e &\leftrightarrow f^{[2]} = fly\text{-}m\text{-}e \end{aligned}$$

The DK encoding, when used in conjunction with iterative deepening on the plan-length bound m is guaranteed to yield a step-optimal plan (Do & Kambhampati 2001). Now, Figure 3 depicts the primal graph $G^{(3)}_{DK}$ of *DK-CSP*$_\Pi(3)$ for our running example. Observe that constraints between variables at adjacent time points in *DK-CSP*$_\Pi(m)$ (i.e., variables

of the form $v_j^{[k]}$ and $v_i^{[k+1]}$) involve *only* neighboring variables within the causal graph. Indeed, $G_{DK}^{(m)}$ corresponds to connected layers of (undirected) causal graphs G_{CG}. Thus, we would expect the tree-width of $G_{DK}^{(m)}$ to be closely related to the tree-width w of G_{CG}, and this is indeed the case.

Lemma 2 *Let Π be a planning problem, and let w be the tree-width of G_{CG}. For any $m \geq 1$, the tree-width w_m of $G_{DK}^{(m)}$ is bounded by $w_m \leq \min\{wm, n\}$. This upper bound is tight for $m > n$, that is, we have $w_m = n$.*

Using Lemma 2, we can immediately provide a structure-aware complexity bound on the "global-length" iterative deepening approach to CSP-based planning.

Theorem 5 *Let Π be a planning problem, w be the tree-width of G_{CG}, and m be the minimal concurrent length of a plan for Π. Then, Π can be solved in time $O(\min\{nm \cdot a^{n+1}, nm \cdot a^{wm+1}\})$, where $a = \max_i\{|A_i|\}$.*

LID/LOP vs. DK/SOP

With these results, we can compare the relative strengths and weaknesses of DK and LID with respect to their time complexity guarantees. We believe that the overall insight holds for other SOP methods as well. We distinguish between a few cases based on the tightness of the causal graph (w), and the step-length of optimal plans (m).

(1) $w = \Theta(n)$**.** This is the case of very dense causal graphs, indicating strong interactions between variables. This case is a-priori unlikely to be favorable for a factored approach, and Theorems 2 and 5 concur.

(2) $w = O(1)$**.** In that case, LID's complexity is exponential in δ, while DK is exponential in $\min\{m, n\}$. Since $\delta \leq m$, LID dominates whenever $\delta < n$. For example, when $m = O(n^2)$, we must have $\delta \geq n$. However, if $m = o(n \log n)$, and local plans are well balanced (recall our discussion of general factorization), we have $\delta < n$. And if $m = o(n)$, then $\delta < n$ for any factorization.

(3) $w = o(n)$. In such case (e.g., $w = \log(n)$), LID complexity is exponential in δw, whereas DK is exponential in $\min\{mw, n\}$. As in case (2), LID is a win when local plan length is not too large in comparison to n/w, e.g., if $w = \log n$ and $\delta = o(n/\log(n))$.

In short, considering scalability in terms of complexity guarantees, we see that LOP scales better than SOP when local plans are not too long (relatively to n), and the causal tree is not too dense, satisfying the relation $w\delta < \min\{wm, n\}$. Similarly, it can be shown that LOP scales better than OP if the domain preserves the relation $w\delta < \min\{m_{op}, n\}$. Intuitively, if the number of factors grows proportionally to the number of problem variables, and the topology of the causal graph and the required local efforts on the factors remain bounded, LOP will scale up. It is then natural to ask whether interesting problems have such features. While this ultimately requires empirical evaluation, we can already point out a few very encouraging indications.

First, upon examination of the standard benchmarks used in recent IPCs, we found[5] that the step-optimal plan length in ***all*** these benchmarks is relatively low, and does not appear to grow faster than n. Second, if one considers the type of oversubscription planning problems recently discussed in the literature (Smith 2004; Benton, Do, & Kambhampati 2005), one sees that many such problems are characterized by the need to accomplish many, relatively independent and simple tasks (e.g., small experiments at different sites). Finally, (Williams & Nayak 1997) describe planning for mechanical systems with many parts possibly contributing to the plan, but only a small number of actions each. We believe that these observations strongly encourage theoretical and empirical analysis of factored planning.

The Implicit Local Encoding

At this point, we are basically done with our theoretical analysis of factored planning and causal graphs. However, we would like to address one pragmatic problem with the explicit sequence-based encoding that we used throughout the paper. This encoding has variables with very large domains. Each variable assignment makes a relatively strong commitment about the nature of the plan. This has no implications regarding our theoretical worst-case analysis, yet in practice, this is usually not a very good idea. Here we would like to address a question that may have arisen in the mind of the reader regarding the possibility of encoding factored planning in a manner more similar to standard planning-as-CSP approaches, where variable values correspond to the execution of a single action, rather than a sequence of actions. The answer to this question is positive, although the encoding is not obvious, a bit involved, and less intuitive.[6]

The new encoding created for the dth iteration of LID is denoted here by $\mathit{CSP}_\Pi(d)$. The variables of $\mathit{CSP}_\Pi(d)$, are $\mathcal{X} = \{v_i^{(j)} \mid 1 \leq i \leq n, 1 \leq j \leq d\}$ with

$$Dom(v_i^{(j)}) = \{(a, t) \mid a \in A_i \cup NOOP_i, 1 \leq t \leq nd\}\,,$$

where, for each $v_i \in V$ with $dom(v_i) = \{\vartheta_1, \ldots, \vartheta_k\}$, $NOOP_i = \{\mathsf{noop}_{i,1}, \ldots, \mathsf{noop}_{i,k}\}$ is a set of value-preserving dummy actions with $\mathsf{prv}(\mathsf{noop}_{i,j}) = \emptyset$, and $\mathsf{pre}(\mathsf{noop}_{i,j}) = \mathsf{post}(\mathsf{noop}_{i,j}) = \vartheta_j$. In short, the value of $v_i^{(j)}$ captures the jth value change of v_i on a plan for Π. The following five types of constraints are imposed. For ease of presentation, in what follows we refer to the action and time-point components of the value of $v_i^{(j)}$ by $\alpha(v_i^{(j)})$ and $\tau(v_i^{(j)})$, respectively.

(a) *Initial state*: Variables of the form $v_i^{(1)}$ can be assigned only to actions that are executable in the initial state.

(b) *Goal state:* Variables of the form $v_i^{(d)}$ can be assigned only to actions that produce/preserve the goal value for v_i (if any.)

[5] For additional closely related observations, see analysis of planning under "canonicity assumption" in (Vidal & Geffner 2006) where each action is assumed to be required at most once.

[6] This is why we decided to stick with the explicit sequence encoding so far.

(c) *Precondition*: If $\alpha(v_i^{(k)}) = a$, then $v_i^{(k-1)}$ must have as its value an action providing/preserving $\mathsf{pre}(a)[i]$.

(d) *Prevail*: For each $(\overrightarrow{v_k, v_i}) \in CG_\Pi$, and each $1 \leq j, l \leq d$, we pose a constraint $\psi_1 \vee \psi_2 \vee \psi_3$ over $\{v_i^{(j)}, v_k^{(l)}, v_k^{(l+1)}\}$, where

$$\begin{aligned}
\psi_1 &\equiv \mathsf{prv}(\alpha(v_i^{(j)}))[k] = \mathsf{u} \\
\psi_2 &\equiv \tau(v_i^{(j)}) < \tau(v_k^{(l)}) \ \vee \ \tau(v_i^{(j)}) > \tau(v_k^{(l+1)}) \\
\psi_3 &\equiv \tau(v_i^{(j)}) > \tau(v_k^{(l)}) \ \wedge \ \tau(v_i^{(j)}) < \tau(v_k^{(l+1)}) \ \wedge \\
&\qquad \mathsf{prv}(\alpha(v_i^{(j)}))[k] = \mathsf{post}(\alpha(v_k^{(l)}))[k]
\end{aligned}$$

Informally, if the action $\alpha(v_i^{(j)})$ is independent of the value of v_k (that is, ψ_1 holds), or $v_i^{(j)}$ gets scheduled either before the lth or after the $(l+1)$th value changes of v_k (that is, ψ_2 holds), then this particular *pair* of value changes of v_k is irrelevant to applicability of $v_i^{(j)}$. Otherwise, $\alpha(v_k^{(l)})$ should provide $\alpha(v_i^{(j)})$ with the value required by the latter.

(e) *Simultaneity*: For each $(v_i, v_k) \in CG_\Pi$, and each $1 \leq j \leq d$, we pose a constraint $\phi_1 \vee \phi_2$ over $\{v_i^{(j)}, v_k^{(1)}, \ldots v_k^{(d)}\}$, where

$$\begin{aligned}
\phi_1 &\equiv \alpha(v_i^{(j)}) \notin A_i \cap A_k \\
\phi_2 &\equiv \bigvee_{1 \leq l \leq d} \alpha(v_i^{(j)}) = \alpha(v_k^{(l)}) \wedge \tau(v_i^{(j)}) = \tau(v_k^{(l)})
\end{aligned}$$

and a similar constraint is posed in the other direction. Informally, if $\alpha(v_i^{(j)})$ is not one of the actions affecting *both* v_i and v_k, then its scheduling does not require any synchronization between v_i and v_k. Otherwise, the action $\alpha(v_i^{(j)})$ should appear similarly scheduled in time in the sub-plans for both v_i and v_k.

Example 3 *Considering our Rocket domain with $d = 3$, a snapshot of the CSP$_\Pi(d)$ encoding is as follows.*

- Initial-state*: Set* $\alpha(f^{(1)}) \in \{\mathsf{noop}_{full}, \mathit{fly\text{-}e\text{-}m}\}, \alpha(r^{(1)}) = \{\mathsf{noop}_{r=e}, \mathit{fly\text{-}e\text{-}m}\}$*, etc.*
- Goal-state*:*
  For A, we have $\alpha(a^{[3]}) \in \{\mathsf{noop}_{a=m}, \mathit{unload\text{-}a\text{-}m}\}$.
- Precondition *between* $r^{(2)}$ *and* $r^{(1)}$*:*

$$r^{(1)} = (\mathit{fly\text{-}e\text{-}m}, t_1) \longrightarrow \Big(r^{(2)} = (\mathit{fly\text{-}m\text{-}e}, t_2) \vee r^{(2)} = (\mathsf{noop}_{r=m}, t_2)\Big) \wedge (t_2 > t_1)$$

- Prevail *between* $r^{(2)}$ *and* $f^{(1)}$*,* $f^{(2)}$*:*

$$\begin{aligned}
&\psi_1 \longleftrightarrow r^{(2)} = (\mathsf{noop}_{r=e}, t_2) \vee r^{(2)} = (\mathsf{noop}_{r=m}, t_2) \\
&\psi_2 \longleftrightarrow \\
&\quad r^{(2)} = (*, t_2) \wedge f^{(1)} = (*, t_1') \wedge f^{(2)} = (*, t_2') \wedge \big(t_2 < t_1' \vee t_2 > t_2'\big) \\
&\psi_3 \longleftrightarrow \Big(r^{(2)} = (\mathit{fly\text{-}e\text{-}m}, t_2) \vee r^{(2)} = (\mathit{fly\text{-}m\text{-}e}, t_2)\Big) \wedge \\
&\quad \Big(f^{(1)} = (\mathit{fuel}, t_1') \vee f^{(1)} = (\mathsf{noop}_{full}, t_1')\Big)
\end{aligned}$$

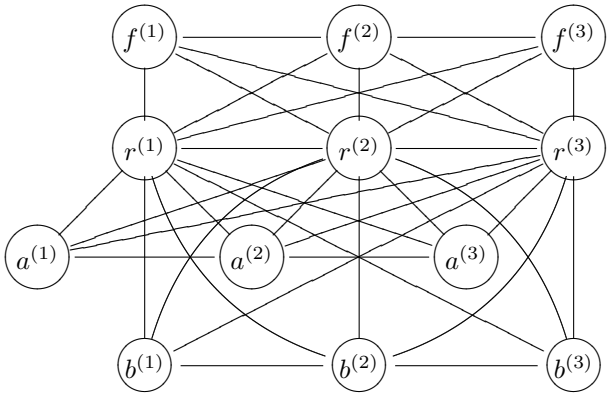


Figure 4: Primal graph $G^{(3)}_{CSP}$ of *CSP*$_\Pi(3)$ for our example.

- Simultaneity *between* $r^{(2)}$ *and* $f^{(1)}$*,* $f^{(2)}$*,* $f^{(3)}$*:*

$$\begin{aligned}
&\phi_1 \longleftrightarrow r^{(2)} = (\mathsf{noop}_e, t_2) \vee r^{(2)} = (\mathsf{noop}_m, t_2) \\
&\phi_2 \longleftrightarrow \xi_{em} \vee \xi_{me}, \text{ where} \\
&\quad \xi_{em} \longleftarrow \Big(r^{(2)} = (\mathit{fly\text{-}e\text{-}m}, t_2) \wedge f^{(1)} = (\mathit{fly\text{-}e\text{-}m}, t_1') \wedge (t_2 = t_1')\Big) \vee \\
&\qquad \Big(r^{(2)} = (\mathit{fly\text{-}e\text{-}m}, t_2) \wedge f^{(2)} = (\mathit{fly\text{-}e\text{-}m}, t_2') \wedge (t_2 = t_2')\Big) \\
&\qquad \Big(r^{(2)} = (\mathit{fly\text{-}e\text{-}m}, t_2) \wedge f^{(3)} = (\mathit{fly\text{-}e\text{-}m}, t_3') \wedge (t_2 = t_3')\Big)
\end{aligned}$$

and ξ_{me} is defined symmetrically.

Lemma 3 LID *based on a sequence of CSP$_\Pi(d)$ problems is a sound and complete planning algorithm. Moreover, given a solvable planning problem Π,* LID *over $\{CSP_\Pi(d)\}_{d=1}^{\infty}$ terminates at iteration δ as in Eq. 2.*

The proof of Lemma 3 is simple, yet technically involved (and thus omitted here), showing that the constraints of *CSP*$_\Pi(d)$ simply imitate these of *SeqCSP*$_\Pi(d)$.

But we would like to establish more than simply the correctness of the implicit encoding. We now prove that the two encodings are equivalent complexity-wise by providing a precise characterization of the tree-width of the primal graph associated with the implicit encoding: Consider the primal graph $G^{(d)}_{CSP}$ of *CSP*$_\Pi(d)$. Not very surprisingly, the structure of this primal graph relates closely to the structure of the causal graph. Specifically, if we have $(v_i, v_j) \in G_{CG}$, then, for $1 \leq k, l \leq d$, we have $(v_i^{(k)}, v_j^{(l)}) \in G^{(d)}_{CSP}$. By exploiting this connection, Lemma 6 provides the precise relation between the tree-width of $G^{(d)}_{CSP}$ and tree-width of G_{CG}, that is, of the causal graph.

Theorem 6 *Let Π be a planning problem, w be the tree-width of G_{CG}, and w_d be the tree-width of $G^{(d)}_{CSP}$. Then, we have $w_d = wd$.*

Conclusion

The idea of divide and conquer through domain decomposition has always appealed to planning researchers. In this paper we provided a formal study of some of the fundamental

questions factored planning brings up. This study resulted in a number of key results and insights. First, it provides a novel factored planning approach that is more efficient than the best previous method of (Amir & Engelhardt 2003). Second, it identifies the domain's causal graph as one of the key parameters in the complexity of factored *and* non-factored planning. Third, the complexity analysis provided enables us to compare between the complexity of standard and factored methods, and provides new classes of tractable planning problems. As we noted, these tractable classes appear to be of genuine practical interest, which has not often been the case for past results on tractable planning. Finally, our analysis helps to understand what makes one factorization better than another, and makes a concrete recommendation on how to factor a problem domain both in presence and in absence of additional domain knowledge.

Future work must examine how well our theoretical insights and new performance guarantees translate into practical performance. However, note that Amir and Engelhardt (2003) have already demonstrated on a certain domain that factored planning can significantly outperform state-of-the-art planners such as FF (Hoffmann & Nebel 2001) and IPP (Köehler & Hoffmann 2000). While the empirical evaluation in (Amir & Engelhardt 2003) is very preliminary, it does indicate that on some non-trivial problems factored planning can be extremely beneficial.

References

Amir, E., and Engelhardt, B. 2003. Factored planning. In *IJCAI'03*, 929–935.

Arnborg, S.; Cornell, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods* 8:277–284.

Bacchus, F., and Yang, Q. 1994. Downward refinement and the efficiency of hierarchical problem solving. *AIJ* 71(1):43–100.

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

Benton, J.; Do, M. B.; and Kambhampati, S. 2005. Over-subscription planning with metric goals. In *IJCAI'05*, 1207–1213.

Brafman, R. I., and Domshlak, C. 2003. Structure and complexity of planning with unary operators. *JAIR* 18:315–349.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

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

Domshlak, C., and Dinitz, Y. 2001. Multi-agent off-line coordination: Structure and complexity. In *ECP'01*.

Erol, K.; Hendler, J.; and Nao, D. S. 1994. HTN planning: Complexity and expressivity. In *AAAI'94*, 1123–1128.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS'04*, 161–170.

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

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI'96*, 1194–1201.

Knoblock, C. 1994. Automatically generating abstractions for planning. *AIJ* 68(2):243–302.

Köehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *JAIR* 12:338–386.

Lansky, A. L., and Getoor, L. C. 1995. Scope and abstraction: Two criteria for localized planning. In *IJCAI'95*, 1612–1618.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *AIJ* 5:115–135.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *ICAPS'04*, 393–401.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. *AIJ* 170(3):298–335.

Williams, B., and Nayak, P. 1997. A reactive planner for a model-based executive. In *IJCAI'97*, 1178–1185.

A Filtering and Decomposition Approach To Optimal Sequential Planning

Stéphane Grandcolas and **Cyril Pain-Barre**
LSIS – UMR CNRS 6168
Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
13397 MARSEILLE CEDEX 20 France
stephane.grandcolas@lsis.univ-mrs.fr, cyril.pain-barre@lsis.org

Abstract

This paper presents the FDP planning system based on the paradigm of planning as constraint satisfaction. FDP integrates consistency rules and filtering and decomposition mechanisms suitable for planning, rather than transform the planning problem into a CSP and make use of an external solver. FDP works directly on a structure related to Graphplan's planning graph. Given a fixed bound on the length of the plan, the graph is incrementally built. Each time the graph is extended, a sequential plan is searched. In the search for a plan different strategies can be employed. Currently, FDP uses a forward-search procedure based on problem decomposition with action sets partitioning.

FDP produces optimal sequential plans. Various techniques are also used to avoid useless processings, in particular to discard redundant sequences of actions and dead-end states.

Empirical evaluation shows that FDP is competitive on many problems, especially compared to other optimal sequential planners.

Introduction

Historically, several translations of planning problems in formalisms of another domain of AI have been attempted. The first one was done in first order logic (Green 1969), at the very beginning of the field of AI planning. Nowadays, most of the new attempts in this direction impose a fixed bound for the plan length which results in a problem that lies in NP. This was first proposed in (Kautz & Selman 1992), where the authors define a SAT encoding of planning problems and use efficient SAT solvers to construct valid plans. The relative success of this approach has led researchers to work on even better SAT encodings, e.g. (Kautz & Selman 1999), or to translate planning problem in other NP-complete formalisms, e.g. Integer Programming (Vossen *et al.* 1999), Constraint Statisfaction (van Beek & Chen 1999).

Some of the encodings of planning problems are based on the well-known *planning graph*. It was proposed in the GRAPHPLAN system (Blum & Furst 1995), and represents important relationships between propositions and actions of the planning problem. During its construction, one can derive additional information about actions and propositions, called *mutex constraints*. The graph and these relations have been successfully represented as SAT encodings in BLACKBOX (Kautz & Selman 1999), and as CSP encodings in GP-CSP (Do & Kambhampati 2001). The system DPPLAN (Baioletti, Marcugini, & Milani 2000) proposes another SAT approach but does not encode the relations of the graph: The actions and the propositions of the planning graph are handled as propositional variables whose truth values are constrained. Each time a variable is assigned a value, several rules are used to *propagate* the change. That may force other variables to be assigned a value. DPPLAN uses a Davis-Putnam-like procedure combined with several possible search strategies for choosing the next variable to assign and its value. Another approach that uses some propagation rules but not based on the planning graph were previously proposed in (Rintanen 1998) to perform non-directional search. In this work, *invariants* are also computed as 2-literal clauses that must hold in some states, and allow the system to further propagate assignments of propositional variables.

More recently, a direct encoding of planning problems as CSP has been proposed (Lopez & Bacchus 2003). It is not based on the planning graph although it captures all of its properties and constraints. This approach allows more constraints to be detected and added, as *additional binary constraints* and *sequence constraints*. Moreover, single valued variable reduction and reasoning about the set of constraints makes it possible to reduce the number of both the variables and the constraints of the problem. The resulting system CSP-PLAN has shown very good performances compared to GRAPHPLAN, GP-CSP and BLACKBOX.

However, CSP-PLAN uses an external solver to search for a solution plan. In this paper, we present the FDP system which handles CSP-like representation of the problem but does not use any external solver. As DPPLAN, FDP integrates consistency rules and filtering and decomposition mechanisms suitable for planning. That allows to totally control the search procedure and the CSP mechanisms and to adapt them for the purpose of planning.

FDP works on a structure that resembles GRAPHPLAN's planning graph without no-ops actions. It is a leveled graph composed of *steps*. A step consists in an *action level* fol-

lowed by a *proposition level*. The structure is incrementaly extended until a solution is found or a fixed bound of the number of steps is reached. The current implementation extends the structure with one step more and the search procedure is forward-chaining with depth-first search, based on problem decomposition with action sets partitioning. Hence, it is basically *Depth-First Iterative Deepening* (Korf 1985) (or IDA^* with admissible heuristic of constant cost 1). This will eventually change in further implementations. First, other heuristics can be employed, that are admissible or not, e.g. (Bonet & Geffner 2001; Hoffmann & Nebel 2001). Furthermore, the underlying consistency rules and filtering procedure make it possible to operate any decomposition anywhere in the structure. Hence, it is possible not to perform solely forward-chaining but also backward-chaining, or bidirectional search, and more generally, undirectional search as all the approaches to planning as constraint satisfaction or satisfiability do, e.g. (Rintanen 1998; Baioletti, Marcugini, & Milani 2000; Lopez & Bacchus 2003).

The current implementation of FDP does not detect unsolvability of problems, as many other similar approaches (Rintanen 1998; Baioletti, Marcugini, & Milani 2000; Lopez & Bacchus 2003). Then, it must be given a fixed bound of plan length in order to stop on unsolvable instances of problems. This weakness of the algorithm will be adressed in future work.

The search procedure is complete. If a solution is found, it is minimal in terms of plan length. On the other hand, the current search procedure of FDP requires that any solution must contain only one single action per step. Hence, solutions returned by FDP are optimal in terms of the number of actions. This feature becomes quite rare for nowadays planners. Some of them perform heuristic search without any guaranty of minimality of solutions (Bonet & Geffner 2001; Hoffmann & Nebel 2001). Others search for minimal solutions in plan length with several actions per step but not in the number of actions, e.g. (Blum & Furst 1995; Do & Kambhampati 2001; Vidal & Geffner 2004). We believe that optimality in the number of actions may be of interest for some applications, in particular if actions are costly. A future development of this work will take into account cost-values of actions for producing minimal plans.

However, it should be noted that non optimal planners (in the number of actions) often scale better and solve larger problems than optimal sequential planners do. Even if we have observed that FDP remains rather competitive on some problems, those planners cannot be compared, as they do not have the same objectives. Empirical evaluation with some optimal sequential planners are presented in the paper, that show that FDP is competitive.

The input language is PDDL with typing and equality. Initial state is interpreted according to the closed world assumption. Given the initial state of a problem and the set of operators of the domain, we construct all the propositions and all the operator instances (i.e. actions) by the application of every operator. Hence, the system only works on propositions and fully instantiated actions.

The paper is organized as follows: first we define the structure the system uses, then we introduce the consistency rules and we present the consistency procedure, next the search procedure is described, and finally experimental results are presented that show the competitiveness of FDP and we conclude with some future developments of this work.

Preliminaries

The purpose of planning is to find a *plan*, which is a set of actions taken in a given set A and an (partial or total) ordering of them, such that the execution in a given *initial state* I of any sequence of the plan's actions according to the ordering achieves given *goals* G.

Many planners returns solution plans of length k that are of the form A_0, A_1, ..., A_k, where each A_i is a subset of A. Such plans mean that every action of A_i must be executed before those of A_{i+1}. Actions of the same set A_i can be executed in any order, even in parallel. In this case, $k+1$ is said to be the *makespan*, i.e. the earliest time to achieve the goals. In the approach we present herein, each A_i will be a singleton, i.e. FDP returns plans as totally ordered sequences of actions. On the other hand, FDP returns optimal plans in terms of the number of actions, what usually is not guaranteed by parallel approaches.

Definition 1 (Planning Problem) *A planning problem $\mathcal{P}$ is a 4-uple (A, I, G, P) where A is a set of actions, I is the initial state, G are the goals, and P is the set of all the propositions that occur in I, G and in A (in positive or negative form).*

The initial state I and the goals G are conjunctions of literals (i.e. possibly negated propositions). The set of the literals is noted L. The set I describes what must be true in the initial state. It is interpreted according to the *closed world assumption*, that is every proposition $p \in P$ that does not appear in I is supposed to be false ($\neg p$ is true in I). The set G describes the literals the plan must achieve, that is literals that must be true in the last state.

Once PDDL-operators have been instantiated, FDP handles classical grounded actions. Every action $a \in A$ is described by its preconditions $\mathsf{pre}(a)$, and its effects $\mathsf{eff}(a)$. $\mathsf{pre}(a)$ is a set of literals that must be true for a to be applicable, and $\mathsf{eff}(a)$ is a set of literals that a makes true.

FDP makes use of special CSP-like representations to find valid plans of a given length, called *fdp-structures*.

Definition 2 (fdp-structure) *Given a planning problem $\mathcal{P} = (A, I, G, P)$, let define a fdp-structure for $\mathcal{P}$ as a 4-uple $\langle k, V_a, V_p, d\rangle$, where:*

- *k is the size of the fdp-structure,*
- *$V_a = \{a_0, \ldots, a_{k-1}\}$ is a set of* action variables
- *$V_p = \{x_{i,p}\}_{0 \leq i \leq k, p \in P}$ is a set of* proposition variables,
- *d is a function which makes correspond variables with their domains:*
  - *to each action variable $a_i \in V_a$ a subset of a denoted A_i,*
  - *to each proposition variable $x_{i,p} \in V_p$ a subset of $\{\text{TRUE}, \text{FALSE}\}$, denoted $D_{i,p}$.*

A fdp-substructure *of the fdp-structure $\langle k, V_a, V_p, d\rangle$ is a 4-uple $\langle k, V_a, V_p, d'\rangle$ such that for each $x \in V_a \cup V_p,\ d'(x) \subseteq d(x)$.*

A proposition variable whose domain is $\{\text{TRUE}, \text{FALSE}\}$ is *undefined*. If v is the value TRUE (resp. FALSE), its *opposite* denoted $opposite(v)$ is the value FALSE (resp. TRUE). For convenience, we will use *virtual* proposition variables. To each proposition variable $x_{i,p}$ corresponds a virtual proposition variable $x_{i,\neg p}$ whose domain is $D_{i,\neg p} = \{opposite(v) \mid v \in D_{i,p}\}$, i.e. $D_{i,\neg p}$ contains the opposites of the possible values for $x_{i,p}$. In the following we will then use (virtual) proposition variables $x_{i,l}$ where l is a literal. Virtual proposition variables will serve as a shortcut, since it will be more convenient to talk about truth value of a literal l than to distinguish between the values TRUE and FALSE of a proposition. Moreover, virtual proposition variables must be considered as "views", since removing a value v from the domain of a (virtual) proposition variable $x_{i,l}$, also removes the value $opposite(v)$ from the domain of the (virtual) proposition variable $x_{i,\neg l}$. In the following, the symbol p stands for propositions, and l stands for literals. >From now on we will consider the problem $\mathcal{P} = (A, I, G, P)$ and we will always refer to fdp-structures for this problem.

A fdp-structure can be viewed as a graph that resembles the well-known GRAPHPLAN planning graph (Blum & Furst 1995). It is a leveled graph that alternates *propositions levels* and *actions levels*. The i-th propositions level, denoted P_i, represents the validity of the propositions at step i: it is the set of the proposition variables $\{x_{i,p}\}_{p \in P}$. The i-th actions level A_i represents the possible values for the action that is applied at step i. Note that a fdp-structure does not contain no-ops actions.

Definition 3 (Valid Plan) *A* valid plan *for the fdp-structure $\langle k, V_a, V_p, d\rangle$ is an assignment θ of the variables in $V_a \cup V_p$ such that:*

1. $\forall a_i \in V_a,\ \theta(a_i) \in A_i$,
2. $\forall x_{i,p} \in V_p,\ \theta(x_{i,p}) \in D_{i,p}$,
3. $\forall l \in I,\ \theta(x_{0,l}) = \text{TRUE}$, *and* $\forall l \notin I,\ \theta(x_{0,l}) = \text{FALSE}$,
4. $\forall l \in G,\ \theta(x_{k,l}) = \text{TRUE}$,
5. $\forall a_i \in V_a, \forall l \in \mathsf{pre}(\theta(a_i)),\ \theta(x_{i,l}) = \text{TRUE}$,
6. $\forall a_i \in V_a, \forall l \in \mathsf{eff}(\theta(a_i)),\ \theta(x_{i+1,l}) = \text{TRUE}$,
7. $\forall l \in L$, *if* $\theta(x_{i,l}) \neq \theta(x_{i+1,l})$ *then* $l \in \mathsf{eff}(\theta(a_i))$ *or* $\neg l \in \mathsf{eff}(\theta(a_i))$.

where $\forall p \in P,\ \theta(x_{i,\neg p}) = opposite(\theta(x_{i,p}))$.

The sequence of the actions of the valid plan (exactly one action per step) is a plan of length k for the problem $\mathcal{P}$. Searching for an optimal plan in the number of actions consists in searching for the shortest $\mathcal{P}$ valid plan.

Consistent fdp-structures

Given a fdp-structure, consistency rules aim to remove possible values of proposition variables or actions that cannot occur in any valid plan. For example an action whose one precondition is not valid should not be considered, and then can be removed without loss of completeness.

These built-in rules capture all the axioms of satisfiability approaches to sequential planning. They correspond roughly to the propagation rules of DPPLAN, with the major difference that DPPLAN is a parallel planner.

Definition 4 (Inconsistent values of proposition variables) *Given the fdp-structure $\langle k, V_a, V_p, d\rangle$, the value* **TRUE** *for the (virtual) proposition variable $x_{i,l}$ (i.e the proposition variable $x_{i,p} \in V_p$ or the virtual proposition variable $x_{i,\neg p}$) is inconsistent if any of the following situations holds:*

1. **(forward persistency)** $0 < i \leq k$, TRUE $\notin D_{i-1,l}$ *and* $\forall a \in A_{i-1},\ l \notin \mathsf{eff}(a)$,
2. **(all actions delete)** $0 < i \leq k,\ \forall a \in A_{i-1},\ \neg l \in \mathsf{eff}(a)$,
3. **(backward persistency)** $0 \leq i < k$, TRUE $\notin D_{i+1,l}$ *and* $\forall a \in A_i,\ \neg l \notin \mathsf{eff}(a)$,
4. **(opposite always required)** $0 \leq i < k, \forall a \in A_i,\ \neg l \in \mathsf{pre}(a)$.

One can note that the fourth rule is original and does not exist in DPPLAN.

Definition 5 (Inconsistent values of action variables) *The action $a \in A_i$ is inconsistent if one of the following situations holds:*

1. **(falsified precondition)** $\exists l \in \mathsf{pre}(a)$ *such that* TRUE $\notin D_{i,l}$,
2. **(falsified effects)** $\exists l \in \mathsf{eff}(a)$ *such that* TRUE $\notin D_{i+1,l}$,
3. **(effect required)** $\exists l \in L$ *such that* TRUE $\notin D_{i,l}$, FALSE $\notin D_{i+1,l}$ *and* $l \notin \mathsf{eff}(a)$.

A value for a proposition variable that is not inconsistent is consistent. An action that is not inconsistent is consistent.

Definition 6 (Consistent fdp-structure) *The fdp-structure $\langle k, V_a, V_p, d\rangle$ is* consistent *if and only if:*

1. *the domain values of the proposition variables and action variables are not inconsistent,*
2. *for each proposition p and for each step i, $D_{i,p}$ is not empty,*
3. *for each step i, A_i is not empty.*

Property 1 (Consistency in a Valid Plan) *Given a valid plan θ for a fdp-structure $\langle k, V_a, V_p, d\rangle$, for each $a \in V_a$ the action $\theta(a)$ is consistent, and for each $x \in V_p$ the value $\theta(x)$ is consistent.*

Proof 1 *Obviously, any inconsistent value or action violates the definition of a valid plan.*

Property 2 (Largest Consistent Substructure) *A fdp-structure S is equivalent to its largest consistent substructure S' if one exists, and this substructure is unique. If no consistent substructure exists then there is no valid plan for S.*

Proof 2 *First note that if a value of a proposition variable or an action is inconsistent in S, then it remains inconsistent in any substructure of S. Then an inconsistent value or action cannot occur within a valid plan and can be removed*

without effect on the set of valid plans for S. Removing iteratively inconsistent values or actions until none exists produces a new structure S' which either contains an empty domain or is consistent and equivalent to S. S' is unique since inconsistent values and actions remain inconsistent in any substructure of S.

Formalizing fdp-structures as constraint satisfaction problems is feasible, making use of n-ary constraints and dynamic CSP (van Beek & Chen 1999; Do & Kambhampati 2001). Since our objective here is just to simplify the structure removing inconsistent values, we make the choice to use specific representation and filtering procedures.

Mutually exclusive propositions and actions Most planning systems support mutual exclusions between propositions: at a proposition level two propositions are mutually exclusive if no valid plan could possibly contain both. Mutual exclusions are propagated through the planning structure (in general a planning graph) extending mutual exclusions to actions and using propagation rules. In these planning systems mutual exclusions help either to construct consistent sets of actions, or to eliminate propositions which are mutually exclusive with instanciated propositions.

FDP does not implement a specific processing for mutual exclusions. The reason is that mutually exclusive actions are useless since FDP produces only sequential plans, and the effects of mutual exclusions of propositions are redundant with FDP inconsistency rules, actually, as for the following mutual exclusions definition: the literals l and l' are mutually exclusive at step i, if one the following situations holds:

1. each action in A_{i-1} either deletes l or l' or both,
2. l is false at step $i-1$ and each action in A_{i-1} that adds l, also deletes l' (or the contrary),
3. l and l' are false at step $i-1$ and no action add both l and l'.

In each case one can prove that mutexes are of no interest. Suppose for example in the case 2 that l' is true at step i. Then, l would not be true at step i since it is mutually exclusive with l'. If the fdp-structure is consistent mutexes are useless: since l' is true at step i, then FALSE $\notin D_{i,l'}$ (and TRUE $\notin D_{i,\neg l'}$). According to definition 5.2, there are no (consistent) action in A_{i-1} that delete l' (i.e have $\neg l'$ in effects). As each action in A_{i-1} which adds l also deletes l', then there are no action in A_{i-1} that add l. Moreover, since TRUE $\notin D_{i-1,l}$, then TRUE $\notin D_{i,l}$ (definition 4.1).

Filtering inconsistent values and actions

Making a fdp-structure consistent consists in removing inconsistent values and actions until none exists or a domain becomes empty. The function **MakeConsistent** is similar to arc consistency enforcing procedures in the domain of constraint satisfaction (Dechter 2003; Mackworth 1977). A queue H contains the *"to remove"* elements. These are 3-uples (p, v, i) representing the value v in $D_{i,p}$ or pairs (a, i) representing the action a in A_i. Until H empty, an element is extracted, the corresponding value or action is removed, and the values or actions which become inconsistent because of this deletion are enqueued in H. FDP inconsistency rules are used to detect these new inconsistencies (**Definition 5** and **Definition 6**). The removals are propagated forward and backward through the fdp-structure.

function **MakeConsistent**(S, H)
in: H a set of values and actions to be deleted,
$S = \langle k, V_a, V_p, d\rangle$ a fdp-structure,
out: TRUE if S is consistent after the propagation of the removals of the facts and actions of H, FALSE if not,

```
1  while H ≠ ∅ do
2      if H contains a 3-uple (p, v, i) then
3          remove (p, v, i) from H,
4          D_{i,p} = D_{i,p} \ {v},
5          if D_{i,p} = ∅ then
6              return FALSE,
7      else if H contains a pair (a, i) then
8          remove (a, i) from H,
9          A_i = A_i \ {a},
10         if A_i = ∅ then
11             return FALSE,
12     insert new inconsistent values and actions in H
13     according to FDP inconsistency rules
14 return TRUE.
```

Propagation stops with failure if a domain becomes empty and **MakeConsistent** returns FALSE. In the other case the procedure stops with the consistent fdp-structure S.

MakeConsistent maintains counters and lists to detect efficiently new inconsistent values and actions. For example at each step i an array of lists of values contains all the values at step i, indexed on the number of actions which delete them. It is then a very low cost to scan the values which are deleted by all the actions at step i.

For each value removal the propagation procedure scans the actions, and for each action removal it scans the proposition variables. In the worst case at each step of the structure, a value for each proposition variable is removed and all but one action are removed. The overall complexity of the function **MakeConsistent** is then $O(k \times |P| \times |A|)$.

Search procedure

To find an optimal plan, FDP starts with a one step fdp-structure, and extends it until a plan is found or a given fixed bound is reached. Each time the fdp-structure is extended, a depth-first search is performed. This ensures the optimality of the solution plan if one exists. FDP employs a *divide and conquer* approach to search for a plan: the structure is decomposed into smaller substructures and the procedure searches recursively each of them. Each time the substructures are filtered so as to detect failures as soon as possible. We have experimented several decompositions such as *enumerating actions*, *assigning proposition variables* or *splitting action sets*. This last one gives the best results: splitting consists in partitionning a set of actions so as to put together actions which own common deletions. Our splitting procedure simply search an undefined proposition variable such that the number of actions that delete it and the number of

actions that do not are as close as possible.

When searching for a plan of length k, FDP uses a fdp-structure S and a set NoGoods. Initially each action set of S is set to A and each proposition variable is undefined. Then, the values which are not in the initial state and the opposites of the goals are removed and a preliminary filtering is performed on S. If S is inconsistent then the search stops with failure, there is no plan of length k. In the other case NoGoods is set to the empty set and the procedure **FDP-search** starts with the consistent structure S. The fdp-structure S is decomposed into two substructures corresponding to the subsets obtained by splitting an action set (line 7). In fact **FDP-search** is a depth first iterative deepening search, since it always chooses the first non singleton action set for splitting, starting from the initial state (lines 4 and 5). Then the current partial plan is extended step by step until the fdp-structure becomes inconsistent or a valid plan is found (since S is consistent, if all the action sets are singletons then no proposition variable is undefined in S and S is a plan).

During the search, each time **FDP-search** encounters a state which does not contain undefined variables and the current call returns failure, this state is memorized as a nogood (lines 16 and 17). The distance $k - i$ from F_{i+1} to the final state is also memorized. Indeed, it is always possible that the state F_{i+1} could be extended to a valid plan if there were more steps (or less, but this situation could not occur in FDP if the nogoods are kept each time the structure is extended: if a smaller valid plan exists from F_{i+1} then it should have been discovered before, while searching a smaller plan). Whenever F_{i+1} will be encountered at a distance less than $k - i$ to the final state the search will be aborted. Memorizing nogoods improves drastically the performances of the search.

function **FDP-search**(i, S)
in: $S = \langle k, V_a, V_p, d\rangle$ a consistent fdp-structure, i a step,
out: TRUE if the structure contains a valid plan, FALSE, otherwise.
1 **if** F_{i+1} does not contain undefined variables **then**
2 **if** $\langle F_{i+1}, k - i\rangle \in$ NoGoods **then return** FALSE,
3 **if** MinSteps$(F_{i+1}) > k$ **then return** FALSE,
4 **while** $|A_i| = 1$ **and** $i < k$ **do**
5 $i = i + 1$,
6 **if** $i = k$ **then return** TRUE, // S is a valid plan
7 $\langle A', A''\rangle =$ SplitActionSet(A_i),
8 $S' = S$,
9 **if** MakeConsistent(S', A'') **then**
10 **if** FDP-search(i, S') **then**
11 **return** TRUE,
12 $S' = S$,
13 **if** MakeConsistent(S', A') **then**
14 **if** FDP-search(i, S') **then**
15 **return** TRUE,
16 **if** F_{i+1} does not contain undefined variables **then**
17 NoGoods = NoGoods $\cup\{\langle F_{i+1}, k - i\rangle\}$,
18 **return** FALSE.

Minimal bound of the number of steps

Anytime a propositional level F_i is completely instanciated, FDP performs a greedy evaluation of a minimal bound of the number of steps to achieve all the unsatisfied goals. The idea is to choose at each of the following steps the action which adds the most unsatisfied goals. In the best case these actions will constitute a valid plan. If the number of unsatisfied goals added by the selected actions is less than the number of unachieved goals, then the current instanciation is rejected (line 3, the function **MinSteps** evaluates the minimal number of steps which are necessary to achieve all the missing goals).

Redundant actions sequences

Since FDP searches sequential plans, it can generate equivalent permutations of "independent" actions and perform as many redundant processings. This constitutes the main defect of sequential planning compared to parallel planning. To cope with this problem, FDP discards sequences of independent actions that do not respect an arbitrary total order on the actions denoted $\prec$.

Definition 7 (Ordered 2-Sequences) *The actions a_1 and a_2 are* independent *if the following situations hold:*

1. $\forall l \in \mathsf{pre}(a_1)$ *then* $l \notin \mathsf{eff}(a_2)$ *and* $\neg l \notin \mathsf{eff}(a_2)$[1],
2. $\forall l \in \mathsf{pre}(a_2)$ *then* $l \notin \mathsf{eff}(a_1)$ *and* $\neg l \notin \mathsf{eff}(a_1)$.

The sequence (a_1, a_2) is an ordered 2-sequence *if either a_1 and a_2 are independent and $a_1 \prec a_2$, or a_1 and a_2 are not independent.*

FDP discards unordered 2-sequences. Besides, it also discards sequences whose actions have exactly opposite effects, i.e such that $\mathsf{eff}(a_1) = \{\neg l \mid l \in \mathsf{eff}(a_2)\}$, as such sequences are useless in a plan.

Property 3 *For any plan $(a_0, \ldots, a_{k-1})$, there exists a permutation of the actions $a_0, \ldots, a_{k-1}$ which is a plan, such that any two successive actions constitute an ordered 2-sequence.*

Proof 3 *It suffices to permute successive independent unordered actions as much as possible in the initial plan. Each transformation produces a plan which is equivalent to the initial plan. When the processus stops any two successive actions form an ordered 2-sequence.*

To implement this property the following rules are added to the definition 5, of inconsistent actions:

4. **(no backward ordered 2-sequence)**
$i > 0$ and $\forall a' \in A_{i-1}$, (a', a) is not an ordered 2-sequence,
5. **(no forward ordered 2-sequence)**
$i < k - 1$ and $\forall a' \in A_{i+1}$, (a, a') is not an ordered 2-sequence.

[1] If a_1 requires a fact which is added by a_2, it is possible in some situations that the sequence (a_2, a_1) must be authorized. Then a_1 and a_2 should not be considered as independent.

Relevant literals and actions

FDP searches optimal sequential plans. Then actions which do not help effectively to achieve the goals are useless and should not be considered. In particular, this is the case of actions of the last step not adding goals. This property can be propagated backwards introducing the notion of *relevant literals and actions* at some steps.

Definition 8 (Relevant literals and actions) *In a given fdp-structure,* relevant actions *and* relevant literals *are recursively defined as follows:*

1. *each goal in G is relevant at step k,*
2. *a literal l is relevant at step i if there exists an action $a \in A_i$ such that l is a precondition of a and a is relevant at step i,*
3. *an action a is relevant at step i if one of its effects is relevant at step $i+1$.*

Actions that are not relevant at a given step can be removed from this step as it could not serve in a minimal solution. The function **MakeConsistent** performs these removals whenever it is possible.

Experimental results

We have compared FDP with some other planners on several problems taken from usual planning benchmarks. They are the problems *mystery* and *mystery-prime*, *hanoi towers*, *xy-world*; problems from IPC3 competition: *depot*, *driver-log*, *freecell*, *satellite*, *zeno-travel*; problems from IPC4 competition: *airport*, *psr-small*, *pipesworld* (*tank-notemp* and *notank-notemp*), *optical*, *philosophers* and *satellite*. In each serie some problems have been rejected: the problems that all the planners solve in less than 0.1 second, and the problems that all the sequential planners (and some parallel planners) could not solve in less than 1,500 seconds. Also, for the remaining problems, we have selected only representative instances in each serie. For example, many instances in the *mystery* and *psr* series have similar results and just some of them are reported here.

As previously said, action sets splitting is the best decomposition among strategies that are proposed in this work. It is then used in both the **fdpN2s** and **fdp2s** versions of FDP. The difference between **fdp2s** and **fdpN2s**, is that the former takes into account the *ordered 2-sequences* and discards unordered sequences of actions. In few problems, the computation and the use of all the permitted 2-sequences is more costly than the benefits they provide. It is the case for example for the *mystery* series, in which there are many actions and many permitted 2-sequences. There can be several reasons: First, the computation of the 2-sequences is polynomial-time in the number of actions. Second, the benefits of the 2-sequences decrease with their number. Third, taking the 2-sequences into account produces a better filtering but then the heuristics used to split the action sets could in some cases give worse results.

Apart from such problems, one can note that **fdp2s** performs better than **fdpN2s**, even if it is not drastically.

We have also reported the experimental results of the following planners: BFHSP (Zhou & Hansen 2004) with a backward search and h3max option (**bfhspbk**), BFHSP with a forward search and h1max option (**bfhspfw**), HSP (Haslum & Geffner 2000) with the command line option `-seq` to generate sequential plans (**hspSeq**), SATPLANNER (Rintanen 2004) a planner based on satisfiability testing that we run with the Siege V4 SAT solver and the `-opt` option to produce optimal sequential plans, DPPLAN (Baioletti, Marcugini, & Milani 2000) which uses propagation rules in a Davis-Putnam-like search procedure, and SATPLAN (Kautz & Selman 1999) that we run with the Siege V4 SAT solver. The last two produce parallel plans whereas the first four produce optimal sequential plans. Some problem instances have resulted in some errors that are different given the planner. They are reported with – (two many operators, parse error, max steps reached, planner killed, time or memory exceeded,...). Time limit was fixed to 1500 seconds. The computation times are summarized for each serie (various domains, IPC3 and IPC4).

One can remark that in general parallel planners perform better than sequential planners, since parallel plans are shorter. In fact, optimal parallel planners and optimal sequential planners are not comparable, as they do not search for the same optimality. However, we have reported the results of DPPLAN and SATPLAN since they serve as a reference. In particular, DPPLAN is a planner that also uses propagation rules and is very related to the work we present in this paper.

It seems that FDP and BFHSP perform better than the other sequential planners. FDP is generally better for problems with a large number of operators but short plans, as the mystery domains. On these problems it seems that a breadth-first search will spend a significant amount of time to compute huge layers of states. On the contrary, when there are few actions and long plans, as in Hanoi and FreeCell, BFHSP planners dominate. On the other hand , very rarely **bfhsp-back** and **bfhsp-forw** have both a good performance whereas **fdp2s** for instance is often better than at least one version of BFHSP.

Finally, the main advantage of FDP is its regularity. Some planners are much more efficient on some problems. For example DPPLAN and **bfhsp-back** are really faster for IPC3 serie. But on other series they have bad results and were stopped with timeout (and in some cases where the processes were not easily killed, the search was finally stopped after more than 10000 seconds) or sometimes stopped with errors. This is rarely the case with FDP which was stopped with timeout only one time on this selection of problems.

Conclusion and perspectives

We described in this paper FDP, a new planner which builds sequential optimal plans. Compared to other optimal sequential planners FDP seems to be rather competitive. Its consistency rules and its decomposition strategies allow to operate backward chaining search or bidirectional search and more generally undirectional search. FDP could be improved with other evaluations of the minimal distance to the goals (Haslum, Bonet, & Geffner 2005) and concurrent

	actions	facts	lengths	mkspan	fdpN2s	fdp2s	bfhspbk	bfhspfw	hspSeq	satplanner	dpplan	satplan
mprime-x-5	3,464	319	11	6	1,500.00	1,500.00	246.77	1,500.00	1,500.00	136.28	2.53	–
mprime-x-7	1,708	426	5	1	0.63	1.10	282.63	13.49	15.95	8.11	1.15	0.18
mprime-x-8	10,206	513	6	5	21.18	75.34	1,500.00	1,500.00	141.52	62.15	–	1.80
mprime-x-21	15,324	838	6	5	15.91	134.43	1,500.00	1,500.00	250.07	–	6.88	–
mprime-x-29	4,872	269	4	4	1.65	9.87	259.66	51.69	63.38	10.74	1.32	1.04
mystery 19	6,521	562	6	6	3.24	21.98	1,139.77	403.15	53.49	59.32	3.28	17.86
mystery 30	4,085	412	9	6	50.42	23.45	392.35	1,500.00	27.12	41.44	3.08	26.06
hanoi 6	166	75	63		3.77	2.73	0.46	0.18	1,500.00	1,500.00	1,500.00	1,500.00
hanoi 7	238	94	127		33.31	26.41	–	–	1,500.00	1,500.00	1,500.00	1,500.00
xy-world f10	200	40	10	1	0.06	0.06	74.08	270.29	–	8.52	0.01	0.18
Sum Various					**1,630.10**	**1,795.30**	**5,395.70**	**6,738.80**	**5,051.50**	**3,326.50**	**3,018.20**	**3,047.10**
depot 2	180	82	15	8	3.78	2.18	0.46	6.35	21.89	3.89	0.08	0.50
Driver log 4	144	63	16	7	56.84	27.77	1.09	113.90	1,500.00	213.38	0.44	0.45
Driver log 5	168	75	18	8	201.92	108.60	27.89	896.25	1,500.00	990.14	0.07	0.65
FreeCell 2	1,144	139	14	8	130.78	110.33	6.27	347.27	1,500.00	1,413.84	31.20	1,500.00
FreeCell 3	1,616	183	18	7	1,500.00	1,283.06	94.22	1,500.00	1,500.00	1,500.00	2.01	1,500.00
satellite 3	188	66	11	6	1.24	0.99	0.21	138.81	3.05	0.93	0.04	–
satellite 4	259	71	17	10	89.18	60.20	4.33	1,500.00	1,500.00	99.15	17.51	–
Zeno 5	376	52	11	5	5.65	3.57	0.41	41.91	–	60.32	0.07	0.46
Zeno 6	392	58	11	5	45.28	27.51	1.96	297.84	–	53.31	0.14	2.65
Zeno 7	408	64	15	6	41.46	30.03	13.77	772.86	–	1,500.00	0.08	1.02
Sum IPC3					**2,076.10**	**1,654.20**	**150.60**	**5,615.10**	**7,524.90**	**5,834.90**	**51.60**	**3,005.70**
Airport 12	203	356	39	21	5.32	3.95	11.49	0.77	2.34	159.55	0.15	1.13
Airport 14	347	493	60	26	744.25	546.11	225.78	35.26	1,500.00	1,500.00	65.32	3.30
PSR 19	163	41	25	15	7.09	6.20	42.79	1.62	1,500.00	25.77	549.18	1.79
PSR 22	112	56	33	25	76.69	145.11	102.42	135.72	1,500.00	106.56	1,500.00	168.86
PSR 25	9,400	58	9	9	31.34	54.96	22.56	126.41	130.75	27.37	–	132.44
PSR 36	343	82	22	16	1,040.47	697.83	128.20	1,500.00	1,500.00	11.72	1,500.00	7.16
PSR 40	762	66	20	15	321.27	278.83	31.39	396.39	1,500.00	11.10	1,500.00	18.09
PSR 46	98	60	34		43.20	74.57	996.85	79.30	1,500.00	254.20	1,500.00	1,500.00
PipesW NT-NT 11	720	211	20		227.19	243.66	720.98	278.85	1,500.00	1,500.00	1,500.00	1,500.00
PipesW NT-NT 13	896	251	16	12	124.12	110.54	62.22	147.93	1,500.00	160.61	287.99	81.64
PipesW NT-NT 21	1,140	292	14	14	18.49	8.58	70.91	104.29	1,500.00	20.53	1,500.00	200.82
PipesW T-NT 5	768	164	8	7	3.37	1.87	–	74.03	231.70	13.90	1.41	3.46
PipesW T-NT 6	768	164	10	7	19.03	12.05	–	301.49	1,500.00	66.38	2.06	9.42
PipesW T-NT 7	2,672	204	8	7	37.01	19.11	–	1,500.00	1,500.00	–	1,500.00	512.50
Optical 1	446	282	36	13	177.80	135.26	–	34.79	1,500.00	1,500.00	0.18	1.74
Philosophers 3	112	120	44	11	141.72	158.56	–	12.33	1,500.00	1,500.00	0.16	0.56
satellite 3	188	66	11	6	1.24	0.99	0.21	165.05	3.03	0.97	0.04	7.57
satellite 4	259	71	17	10	88.13	60.30	3.90	1,500.00	1,500.00	85.75	19.48	165.89
Sum IPC4					**3,107.70**	**2,558.40**	**2,419.70**	**6,394.20**	**21,367.80**	**7,544.40**	**11,425.90**	**4,316.40**

Figure 1: CPU times for different planners on a serie of selected problems (times are in seconds on a Linux computer with a Pentium 3 Ghz processor, and 1 Go RAM).

bidirectional searches which could cooperate through valid or invalid states. The lack of termination criterion will be also adressed in future work. Finally FDP could be extended to handle valued actions and to compute plans of minimal costs. Also, planning with ressource will be a matter of development.

References

Baioletti, M.; Marcugini, S.; and Milani, A. 2000. Dpplan: An algorithm for fast solutions extraction from a planning graph. In *AIPS*, 13–21.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Dean, T., ed. 1999. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Morgan Kaufmann.

Dechter, R. 2003. *Constraint Processing.* Morgan Kaufmann, San Francisco.

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

Green, C. C. 1969. Application of theorem proving to problem solving. In *IJCAI*, 219–240.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *AIPS*, 140–149.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In Veloso, M. M., and Kambhampati, S., eds., *AAAI*, 1163–1168. AAAI Press AAAI Press / The MIT Press.

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

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *ECAI*, 359–363.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In Dean (1999), 318–325.

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a CSP. In Gottlob, G., and Walsh, T., eds., *IJCAI*, 954–960. Morgan Kaufmann.

Mackworth, A. 1977. Consistency in networks of relations. In *Artificial Intelligence*, 8:99–118.

Rintanen, J. 1998. A planning algorithm not based on directional search. In *KR*, 617–625.

Rintanen, J. 2004. Evaluation strategies for planning as satisfiability. In de Mántaras, R. L., and Saitta, L., eds., *ECAI*, 682–687. IOS Press.

van Beek, P., and Chen, X. 1999. Cplan: A constraint programming approach to planning. In *AAAI/IAAI*, 585–590.

Vidal, V., and Geffner, H. 2004. Branching and pruning: An optimal temporal pocl planner based on constraint programming. In McGuinness, D. L., and Ferguson, G., eds., *AAAI*, 570–577. AAAI Press / The MIT Press.

Vossen, T.; Ball, M.; Lotem, A.; and Nau, D. S. 1999. On the use of integer programming models in ai planning. In Dean (1999), 304–309.

Zhou, R., and Hansen, E. A. 2004. Breadth-first heuristic search. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *ICAPS*, 92–100. AAAI.

On State Management in Plan-Space Planning from CP Perspective

Pavel Surynek

Charles University
Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
surynek@ktiml.mff.cuni.cz

Abstract

In this paper we address the problem of encoding of plan-space planning problems as a CSP. We propose a constraint model for expressing plan-space partial plans. As an enrichment of the notion of partial plans we propose a world state management within the partial plan. We incorporate the world state management into our CSP model too. Next we propose a specialized algorithm for solving the CSP model. The algorithm builds the constraint model dynamically as conflicts are resolved in the model. The performed preliminary experiments showed the usefulness of state management in the constraint model. The usage of state management allows the additional search space pruning compared to the model without states.

Introduction

Planning is a widely studied topic of artificial intelligence. The importance of studying planning arises from needs of real-life applications such as industrial automation, transportation, robotics and other braches (Nau, Regli, Gupta, 1995). The research in planning is also motivated by the needs of researches in other areas. The most prominent example from the above areas is space exploration where autonomous spacecrafts (Bernard et al., 1998) and vehicles (Ai-Chang et al., 2004) are successfully used. But autonomous devices can be used in many other situations both in science and real-life. The autonomous behavior is controlled by planning techniques and algorithms in many of these cases.

From the traditional view of planning, the planning problem is posed as finding of a sequence of actions which transform a specified initial state of the planning world into a desired goal state of the world (Allen, 1990). The limitation is that only actions from a set of allowed actions can be used. The action typically makes a small local change of the world in which the task takes place.

There are several paradigms for solving planning problems. We deal with so called *state-space* planning (Fikes, Nilsson, 1971) and *plan-space* planning (Sacerdoti, 1975) in this paper. The state-space planning represents a more traditional approach for solving planning problems while the plan-space planning is a newer method, which is currently not so spread.

Although the plan-space planning provides a flexible and intuitive paradigm for solving planning problems, it is less competitive to the recent traditional state-space planning techniques in situations where heuristics play a role. As it is evidenced by the results of the 2004 International Planning Competition, several successful heuristic based planning algorithms (*HSP*, *TP-4*) use state-space approach (Haslum, Geffner, 2001; Bonet, Geffner, 2001). The most significant drawback of plan-space planning is caused mainly by the fact that domain specific heuristics are often based on extracting information from the states of the planning world (Ghallab, Nau, Traverso, 2004). The extraction of information cannot be done in straightforward way within the plan-space framework. On the other hand, another successful planner (*CPT*) in IPC 2004 uses ideas from plan-space approach and from constraint programming (Vidal, Geffner, 2004). This fact was one of our motivations to deal with plan-space planning approach.

The classical state-space planning is based on the notion of an evolution of states of the planning world by planning operators. A given world state is changed by planning operators into new ones. A state based solving algorithm for planning problems performs this evolution in some kind of a controlled way until a required goal state of the planning world is reached. The states of the planning world are explicitly available along the whole resolution process.

Algorithms for solving planning problems within the plan-space paradigm work in a completely different way. They usually refine so called *partial plan* until a certain level of consistency is enforced. The partial plan contains a set of *planning operators* and a set of precedence and binding relations over these operators which the final plan has to satisfy. When the partial plan becomes consistent it is ensured that a valid plan (sequence of fully instantiated planning operators, i.e. *actions*) can be extracted from it. The main difference from state-space planning is the lack

of explicit planning world states during the resolution process. This fact represents the key obstacle in adaptation of domain specific heuristics for plan-space planning paradigm.

Our work is motivated by a number of successful attempts to use constraint satisfaction techniques in planning (Nareyek et al., 2005). We want to propose a framework in which the complete planning problems can be expressed. The design of the framework should allow the use of advanced constraint satisfaction techniques. Our goal is also to preserve world states to be directly accessible along the reasoning process. The explicit presence and accessibility of the world states will allow us to reduce uncertainty in the framework. Moreover, when we have explicit world states, the domain specific heuristics can be adapted more easily.

Many approaches exploiting constraints in planning are based on *Graphplan* algorithm (Blum, Furst, 1997). Graphplan-based algorithms use some kind of compression of many world states into a single structure. Another approach represents *CPlan* algorithm (Van Beek, Chen, 1999). The CPlan algorithm uses state-space based encoding of the whole planning problem up to a certain number of steps. Compared to Graphplan-based algorithms the CPlan algorithm provides an easier access to the states of the planning world along the resolution process. It is enabled by direct encoding of the world states into the model.

Our work is primarily inspired by the CPlan algorithm. But contrary to CPlan we are trying to use state encoding within the plan-space framework. More precisely, we address a problem of integration of state management into the plan-space framework. We formulate partial plans as a *dynamic constraint satisfaction problem* where states of the planning world are also modeled. We propose a specialized search algorithm for this model. Our search algorithm dynamically changes the constraint model according to the applicable refinements until a partially consistent (in our case *arc-consistent*) partial plan is obtained. Then the algorithm continues with trying to extract a valid plan from the consistent partial plan while it is still applying refinements if it is necessary.

The state management in the constraint model of the partial plan allows the search algorithm to reduce uncertainty in the model and to deduce more information about the precedence relations in the partial plan. This leads into a stronger search space reductions and the final fully consistent partial plan is found earlier. The presence of explicit planning world states in the constraint model can be utilized for an easier integration of domain specific heuristics into the search algorithm.

The paper is organized as follows. State-space and plan-space planning paradigms are introduced in the first section. Then we describe how the constraints are used in plan-space planning and the details of our constraint model. Having the constraint model, we design a specialized search algorithm for the model subsequently. Finally we show some empirical results.

State-Space and Plan-Space Planning

The classical formalism for stating planning problems is based on a first order language with finitely many predicate and constant symbols, and infinitely many symbols for variables. There are no function symbols in the language (Ghallab, Nau, Traverso, 2004).

The classical statement of a *planning problem* is a triple $P=(O,s_0,g)$, where s_0 is an *initial state* of the planning world, g is a *goal condition* and O is a finite set of *planning operators*. The initial state s_0 is expressed as a finite set of ground atoms. It is interpreted as a set of propositions that hold in the planning world at the beginning. The goal condition g is a finite set of literals. Similarly, it is interpreted as a set of proposition we want to make true (compared to initial state we allow negative propositions). Finally, O is a finite set of planning operators, through which it is possible to locally change the world states. The planning operator is a triple $o=(name(o),precond(o),effects(o))$, where $name(o)$ denotes the name of the operator, $precond(o)$ is an operator precondition and $effects(o)$ is an effect of the operator. Precondition is modeled as a finite set of atoms which must hold in the world state before the operator can be applied. The effect of the operator is modeled as a finite set of literals that hold in the state after the operator is applied. The task is to find a sequence of ground planning operators (*actions*) such that it transforms the given initial state into a state satisfying the given goal condition. We call this sequence a *plan*.

The required sequence of actions is searched directly within the state-space planning. The search process starts from the initial or from the goal state and builds the sequence of actions by adding a new action at the end of the already finished partial sequence of actions. The current state of the world corresponding to the application of the partial sequence of actions on the initial state or on the goal state is maintained along the whole search process.

In plan-space planning, we work with so called partial plans. A *partial plan* is a tuple $\pi=(A,\prec,B,L)$ (Ghallab, Nau, Traverso, 2004), where A is a finite set of partially instantiated planning operators present in the partial plan, $\prec$ is a finite set of precedence constraints on the elements of A, B is a set of binding constraints, which bind variables appearing in the planning operators in A, and L is a set of *causal links*. A causal link is a relation of the form $a_i \xrightarrow{p} a_j$, where $a_i,a_j \in A$ and a_i has an effect p, which is required as a precondition for a_j.

CSP and Plan-Space Planning

The structure of the partial plan is very close to the structure of constraint satisfaction problem. Both structures use the notion of a constraint between a set of objects.

A *constraint satisfaction problem* (CSP) is a triple (X,D,C) (Dechter, 2003), where X is a finite set of variables, D is a finite domain of values for the variables from X and C is a finite set of constraints over the vari-

ables from X. The constraint can be an arbitrary relation over the elements of the domains of its variables. Having a constraint satisfaction problem the task is to find an assignment of values to the variables such that all the constraints are satisfied.

In our framework for plan-space planning we model partial plans as constraint satisfaction problems. Actions occurring in the partial plan are modeled as CSP variables whose domains are sets of possible moments of execution of the corresponding action. In the following sections we will denote these variables as *time variables*. With time variables it is easy to translate temporal precedence relations directly into constraints of the constructed CSP model. Since we will be working only with actions (ground operators) we do not need to translate binding relations into constraints of our CSP model (note that we would also need other types of variables if we want to model partially instantiated operators). Note that this kind of constraint satisfaction problem can be solved in polynomial time (Dechter, Meiri, Pearl, 1991). Such simplified problem does not exploit the full generality of CSP which is NP-hard in general. However, our constraint model will be extended with other types of variables and constraints in the following sections.

Since the standard resolution process over the partial plan changes the structure of the plan, we also need to reflect this dynamicity in our CSP model. Typically plan-space algorithms insert new actions and relations into the partial plan as the search proceeds. We can model these changes as incremental additions of variables and constraints into and from our CSP model.

Partial Plans as an Incremental Dynamic CSP

A *dynamic constraint satisfaction problem* (DCSP) (Dechter, 1988) is a sequence of constraint satisfaction problems $P_0, P_1, P_2, \ldots, P_\alpha, \ldots$, where each problem is a result of a modification of the preceding one. The allowed modifications are constraint addition and retraction, and variable addition and removal. The evolution of the dynamic problem is controlled by outside mechanism, for example by the interaction with the user. The controlling mechanism is the solving algorithm itself in our case with partial plans. All the static problems from the sequence must be solved to solve the whole dynamic problem.

Since we are doing only incremental changes in our model we do not need the general dynamic constraint satisfaction problem as given above but only an incremental restriction of DCSP. We also do not follow the above concept exactly in the sense of finding a solution. It is sufficient to solve only the last constraint satisfaction problem from the sequence of the dynamic problem.

Constraint Model

Our constraint model is always built for a limited maximum length of the resulting sequence of actions. Let l be this length. First we will show how actions are stated within the model. Let us consider a ground operator $a = (name(a), precond(a), effects(a))$. For the action a we introduce a time variable $time_a = \{1,2,\ldots,l\}$ into the constraint model*. Semantically the value of the time variable describes the time of the execution of the corresponding action in the final plan. When we are working with non-durative actions only, the value of the corresponding time variable determines the position of the action in the resulting plan. By using time variables we can cover the presence of actions in the partial plan. We define the following bounds for each time variable:

Earliest execution time of action a: $et(a) = minimal\ element\ in\ the\ current\ domain\ of\ time_a$

Latest execution time of action a: $lt(a) = maximal\ element\ in\ the\ current\ domain\ of\ time_a$

Incorporation of States into the Model

For the purposes of more compact representation within the constraint satisfaction problem we require that preconditions and effects of the action are expressed functionally using state variables. More specifically instead of saying that some proposition about some object is true in a given world state, we say that the function describing corresponding property of the object takes a certain value in the given world state. The situation is explained in the following examples. In the example 1 a classical representation of an action is showed.

Example 1. Consider an action that moves robot_1 from location_A to location_B, where robot, location_A and location_B are constants with obvious meaning. We also use a binary predicate *at* saying that the first argument is at the place specified by the second argument. The classical representation of the move action is following:

$$(move-robot\_1-location\_A-location\_B;\ precondition = \{at(robot\_1, location\_A)\};\ effects = \{at(robot\_1, location\_B), \neg at(robot\_1, location\_A)\}).$$

To express the move action functionally we need a function $location : robots \times S \longrightarrow locations$, where *robots* and *locations* are sets of constants for which $robot\_1 \in robots$ and $\{location\_A, location\_B\} \subseteq locations$, and S is the set of states. Let us call this function a *state variable function*. The following example shows the action represented using state variable function.

Example 2. Representation of the action moving robot_1 from location_A to location_B using state variable function.

$$(move-robot\_1-location\_A-location\_B;\ precondition = \{location(robot\_1) = location\_A\};\ effects = \{location(robot\_1) = location\_B\}).$$

* We allow a certain action to appear multiple times in the partial plan.

With each action in the partial plan we associate a set of *precondition variables* and a set of *effect variables*. Together we will call these variables *world state variables*. The set of precondition variables contains one variable for each state variable function describing properties of a certain object in our planning world. Similarly for the set of effect variables, this set also contains one variable for each state variable function in the constraint model. The domains of precondition and effect variables are the same as the ranges of corresponding functions. The situation is explained in more details in the following example.

Example 3. Again consider the action that moves robot_1 from location_A to location_B. In addition we have a new constant robot_2 in this example. There is also the state variable function $location : robots \times S \longrightarrow locations$, where $\{robot\_1, robot\_2\} \subseteq robots$ and $\{location\_A, location\_B\} \subseteq locations$. We introduce following state variables into the constraint model:

$$precond^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A, location\_B, \ldots\}$$
$$precond^{move-robot\_2-AB}_{location(robot\_2)} = \{location\_A, location\_B, \ldots\}$$
$$effect^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A, location\_B, \ldots\}$$
$$effect^{move-robot\_2-AB}_{location(robot\_2)} = \{location\_A, location\_B, \ldots\}.$$

The set of precondition variables describes the world state before the corresponding action is applied. Similarly the set of effect variables describes the world state after the application of the action. The initial state and the goal condition are handled as special actions.

How the Model is Constrained

The above simple collection of variables is sufficient to describe all the objects and possible transitions that can appear within the modeled partial plan. But it is useless without the appropriate set of constraints that would forbid incompatible tuples of values in the domains of variables. A lot of constraints arise directly from the definition of partial plans.

State Transition Constraints. The first type of constraints we introduce into our model ensures that state transitions are reflected in the world state variables. Consider an action and corresponding world state variables that represents the world state before and after the execution of the action. Suppose that the action requires the state variable function f to take the value x before the action can be executed. Then the state transition constraint must ensure that the precondition world state variable corresponding to the state variable function f takes the value x. Next, suppose that the action changes the value of the state variable function f to the value y. Then the state transition constraint ensures that the effect world state variable corresponding to the state variable function f takes the value y. Since these constrains are unary, the information contained in them can be directly encoded into domains of world state variables. The following example shows encoding of state transitions into the variable domains.

Example 4. Let us have the same action and the same constants as in the example 3. We can restrict the domains the world state variables as follows:

$$precond^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_A\}$$
$$effect^{move-robot\_1-AB}_{location(robot\_1)} = \{location\_B\}.$$

The world state variables, whose state variable function does not appear in preconditions or in the effects of the actions, remain unchanged.

Frame Axiom Constraints. State variable functions that do not occur in the effects of the particular action do not change its value after execution of this action. This fact is not implied by the state transition constraints and must be ensured separately. Generally for this reason so called *frame axioms* are used. In our constraint model we also use constraints for encoding frame axioms. The frame axiom constraint ensures that a certain precondition world state variable and the corresponding effect world state variable have the same values. The frame axiom constraints are added for all pairs of precondition and effect variables whose state variable function is invariant with respect to the corresponding action.

Example 5. Let us have the same action and the same constants as in the example 3. The frame axiom constraint is the following:

$$precond^{move-robot\_2-AB}_{location(robot\_2)} = effect^{move-robot\_2-AB}_{location(robot\_2)}.$$

Suppose that there is a frame axiom constraint for action a and state variable function f, then the propagator for the constraint is:

$$precond^a_f \leftarrow precond^a_f \cap effect^a_f$$
$$effect^a_f \leftarrow precond^a_f \cap effect^a_f.$$

Unique Action Time Constraints. Each action in the partial plan has to be executed at a unique time. Our constraint model is designed to produce a plan as a totally ordered sequence of actions, so this is a natural requirement. Another reason for this requirement is that we want to reduce uncertainty in the model as much as possible. It is important for the next type of constraints which are used to manage evolution of states. The unique action time constraint is modeled as *allDifferent* constraint over time variables. Suppose that there are actions $a_1, a_2, \ldots, a_k$ in the partial plan. Then we introduce a constraint $allDifferent(time_{a1}, time_{a2}, \ldots time_{ak})$ into the constraint model. The propagator for the *allDifferent* constraint is described in (Régin, 1994).

State Sequencing Constraints. As the current domains of time variables are narrowed during the search process, there arise situations when it is necessarily true that two actions must be executed consecutively. More specifically, an action must be performed right after another

action if all the other actions are scheduled before or after this pair of actions.

Consider that there are actions a_i and a_j in the partial plan and it is known that these two actions must be executed consecutively in the resulting plan. Without loss of generality suppose that a_j must be executed right after a_i. Then it is possible to post a constraint that binds precondition world state variables belonging to the action a_j to effect world state variables belonging to the action a_i. This idea can be used conversely too. Consider that it is known that effect world state variables of one action cannot be bind to the precondition world state variables of the other action. In such situation we can conclude that these two actions cannot be executed consecutively and it is possible to post a constraint into the model that ensures this fact.

Now we can propose a constraint that realizes the above statements. We will call it a *state sequencing constraint*. It comprises a condition enforcing the equality of effect world state variables and precondition world state variables when the corresponding two actions must be executed consecutively. And next it comprises a condition which does not allow the execution of one action right after the other action if its precondition and effect world state variables are not compatible. The situation is explained in the following example.

Example 6. Let us have an action that moves robot_1 from location_A to location_B and an action that moves robot_1 from location_B to location_C. Then the state sequencing constraints look like as follows:

if $time_{moveBC} = time_{moveAB} + 1$ **then**
$effect^{moveAB}_{location(robot\_1)} = precondition^{moveBC}_{location(robot\_1)}$

if $time_{moveAB} = time_{moveBC} + 1$ **then**
$effect^{moveBC}_{location(robot\_1)} = precondition^{moveAB}_{location(robot\_1)}$.

Since the right hand side of the conditional statement of the second constraint cannot be satisfied, the action *moveAB* cannot be executed right after *moveBC* action.

The propagator for state sequencing constraint is similar to that of frame axiom constraint. Suppose that there are actions a_i and a_j in the partial plan, and state variable function f. Then the propagator is:

if $time_{aj} = time_{ai} + 1$ **then**
$effect^{ai}_f \leftarrow precondition^{aj}_f \cap effect^{ai}_f$
$precondition^{aj}_f \leftarrow precondition^{aj}_f \cap effect^{ai}_f$

if $time_{ai} = time_{aj} + 1$ **then**
$effect^{aj}_f \leftarrow precondition^{ai}_f \cap effect^{aj}_f$
$precondition^{ai}_f \leftarrow precondition^{ai}_f \cap effect^{aj}_f$

if $effect^{ai}_f \neq precondition^{aj}_f$ **then**
propagate $time_{aj} \neq time_{ai} + 1$

if $effect^{aj}_f \neq precondition^{ai}_f$ **then**
propagate $time_{ai} \neq time_{aj} + 1$.

The reason for having state sequencing constraints is not for maintaining world states itself. The whole state management is intended to work reversely. The state sequencing constraint can discover that two actions cannot be consecutive. If we add the fact that the maximum plan length is usually very tight this mechanism allows a significant pruning of current domains of time variables. It is preferred to post as many as possible state sequencing constraints into the model and therefore it is also important to have the above unique action time constraints.

Precedence Constraints. Until now we did not reflect causal links in our constraint model. If there is a causal link $a_i \xrightarrow{p} a_j$ in the partial plan, it is necessary that the action a_i is executed before the action a_j. This relation is simply expressed through a *precedence constraint* between corresponding time variables:

$time_{ai} < time_{aj}$.

The propagator for precedence constraint is the following:

$lt(a_i) \leftarrow lt(a_j) - 1$
$et(a_j) \leftarrow et(a_i) + 1$.

Threat Resolving Constraints. A *threat* for a causal link $a_i \xrightarrow{p} a_j$ is an action in the partial plan that changes the value of a state variable function to the value that is not compatible with the effect p of the action a_i. To resolve the threat it is necessary to ensure that it is executed before both a_i and a_j or after both a_i and a_j. Since the actions a_i and a_j are already ordered by precedence constraint is sufficient to ensure the execution of threat action before a_i or after a_j. Suppose that a_k is a threat for causal link $a_i \xrightarrow{p} a_j$. The corresponding *threat resolving constraint* is:

$(time_{ak} < time_{ai}) \vee (time_{aj} < time_{ak})$.

A propagator for threat resolving constraint is following:

if $lt(a_i) < et(a_k)$ **then**
$lt(a_j) \leftarrow lt(a_k) - 1$, $et(a_k) \leftarrow et(a_j) + 1$

if $et(a_j) > lt(a_k)$ **then**
$lt(a_k) \leftarrow lt(a_i) - 1$, $et(a_i) \leftarrow et(a_k) + 1$.

The Utility of State Management in the Model

The proposed constraint model is built dynamically as the solving algorithm proceeds. During solving process domains of variables in the constraint model are narrowed. It is the standard process how the constraint satisfaction problems are solved.

The narrowing of variable domains results into discovering of new conflicts in the partial plan (for example a threat or unsatisfied precondition of the already present action is discovered). The conflicts are resolved by the extension of the model with new variables and constraints. This may prolong the resulting plan. The extension of the model is forbidden when the maximum allowed length l of the resulting plan is reached. The final

goal of a solving algorithm is to find the total ordering of actions in the plan.

The total ordering of actions in the plan is found when the domains of all the time variables become singleton. In order to reach this we want to prune the domains of time variables as much as possible. The state management in the constraint model helps the search algorithm to discover additional inconsistent values in the domains of time variables and to rule them out.

The situation when the state management is useful is showed in figure 1. Note the propagation of effects in the chain of actions through frame axiom constraints. If there would not be the action that loads the package X at location A in the partial plan, the remaining two actions can be executed consecutively.

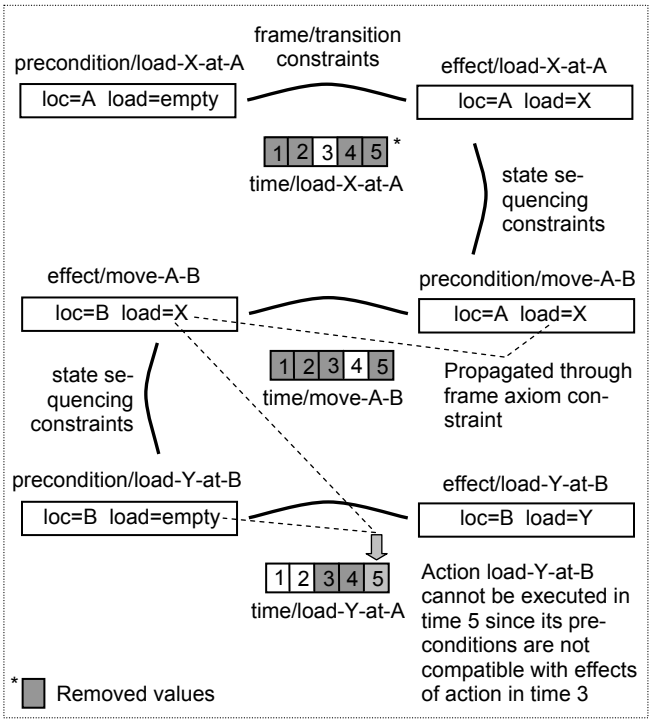


Figure 1: Constraint propagation through world state variables

Algorithm

In this section we describe an algorithm for solving the proposed constraint model. Our algorithm is a combination of standard backtracking based labeling algorithm for solving constraint satisfaction problems and a special resolution mechanism which dynamically extends the constraint model when a flaw in the partial plan is discovered. Along the whole search process the constraint model is maintained in an arc-consistent state. We are using an incremental version of a classical AC-3 algorithm (Mackworth, 1977).

In each step of the algorithm there are two possibilities what can be done. If the constraint model involves a flaw (the flaw in constraint model directly corresponds to a flaw in modeled partial plan) the algorithm can repair the flaw by extending the constraint model with new variables and constraints. The second possibility is to perform a labeling step. The labeling step consists in addition of a labeling constraint. The labeling constraint selects a value for a selected variable and distributes the original constraint satisfaction problem (Schulte, 2002) into two simpler sub-problems. The labeling constraint is positive in the first sub-problem (the selected variable must take a selected value) and negative in the second sub-problem (the selected variable must take the different value than selected). If one of the sub-problems is solved then this solution is also a solution of the original problem.

The flaw resolution step is preferred. Thus if there is a flaw in the constraint model the algorithm would try to repair it and not to label a variable.

In both cases the arc-consistency of the constraint model is restored. The maintaining arc-consistency during the search allows an earlier detection of inconsistencies.

Flaw Resolution Step

The flaw is an *open goal* or a threat. From the previous sections we already know how to resolve a threat. The treat is repaired by posting an appropriate threat constraint into the constraint model.

An open goal is a precondition of an action for that there is no causal link in the partial plan. This means it is not decided how to satisfy the precondition. Consider that the precondition p of the action a_j is an open goal. There are two possibilities how to resolve this open goal. If there is an action a_i already present in the constraint model that has the unsatisfied precondition as its effect. We can simply create a new causal link $a_i \xrightarrow{p} a_j$. After creation of a new causal link a new precedence constraint $time_{ai} < time_{aj}$ is added to the model. Although causal links are explicitly included in our constraint model via precedence constraints, it is more convenient to maintain current causal links in a separate set. Every newly created causal link has to be inserted into this set. The set of causal links can then be used during both kinds of flaw resolution.

Another way how to resolve the open goal is to add a new action into the constraint model that supports the open proposition p of a_j. The addition of a new action into the constraint model is carried out by adding a time variable and corresponding precondition and effect world state variables. It is necessary to insert appropriate set of constraints too.

Consider that we want to add an action a_i. It must have p as an effect. A time variable $time_{ai}$ and world state variables $precondition_f^{ai}$ and $effect_f^{ai}$ for all the state variable functions f are added into the model. State transition constraints are encoded directly into the domains of world state variables. Next we add frame axiom constraints for all the pairs of $precondition_f^{ai}$ and $effect_f^{ai}$ variables, where state variable function f is not affected by the action a_i. The unique action time constraint is extended over the $time_{ai}$ variable. Here a dynamic ver-

sion of *allDifferent* constraint for backtracking based environments can be used (Barták, 2003). The state sequencing constraint is added for all new pairs of actions, i.e. for all pairs of actions (a_i, a_k) where $k \neq i$ a state sequencing constraints that bind $precondition_f^{ai}$, $precondition_f^{ak}$, $effect_f^{ai}$, $effect_f^{ak}$, $time_{ai}$ and $time_{ak}$ variables for all the state variable functions f are added. Finally the precedence constraint $time_{ai} < time_{aj}$ is added.

Labeling Step

The labeling step corresponds to the standard variable labeling from CSP solving algorithms. In each labeling step a variable and a value from its current domain are selected. An ordering heuristics are used for selection of the variable and the value. Currently we use heuristics that select the variable with the smallest current domain and then the first value from its domain is selected for labeling. Let us suppose that variable x and value v is selected for labeling. Then constraints $x = v$ and $x \neq v$ are used to distribute the problem.

The deterministic implementation of the algorithm should use a special ordering heuristics to determine the most promising order of threats and open goals. The heuristics should be also used for selecting the best refinements. However, in our testing implementation we did not use any heuristic for resolution steps. Threats and open goals are tried in the order as they were discovered.

Figure 2 summarizes the whole search algorithm.

```
SolveCSPModel((V,C), I)
(V,C) ← EnforceArcConsistency((V,C))                    } arc-consistency
if exists x∈V with empty current domain then
| return FAILURE;
threats ← GetThreat((V, C))                             } threat resolution
if threats ≠ Ø then
| non-deterministically choose a
|   threat t∈ threats
| (V,C) ← ResolveThreat(t, (V,C), I)
| return SolveCSPModel((V,C), I)
else
| openGoals ← GetOpenGoals((V,C))                       } open goal resolution
| if openGoals ≠ Ø then
| | non-deterministically choose an open
| |   goal g∈openGoals
| | (V,C) ← ResolveOpenGoal(g, (V,C), I)
| | return SolveCSPModel((V,C))
| else
| | if all the variables in V are assigned then        } labeling
| | | return (V,C)
| | select a not yet assigned variable x∈V
| |   according to the variable ordering heuristic
| | select a value v from the current
| |   domain of x
| | non-deterministically choose a labeling
| | constraint c∈ {x=v, x≠v}
| | (V,C) ← (V,C∪{c})
| | return SolveCSPModel((V,C), I)
```

Figure 2: Search algorithm for solving partial plan constraint model

Preliminary Empirical Results

We have implemented the proposed constraint model and search algorithm in C++ in order to examine its properties. The goal of our evaluation is to examine the benefit of state management within the constraint model. We have compared how the algorithm works on the model with and without variables and constraints for managing states. In our preliminary experiments we have used a naive implementation of the *allDifferent* constraint. We intend to use a dynamic *allDifferent* constraint as it is proposed in (Barták, 2003) in the future.

The algorithm was tested on a simple planning problem. There were several packages and locations in the testing planning problem. There were also one or more transporters to transport packages from one location to other location. The transporter can move between locations and can load just one package. The goal was to transport packages to selected locations. We have created several testing instances of our transportation planning problem. These instances differ in the minimum plan length and in number of goals that has to be satisfied (if it is required to transport two packages there are two goals).

Our search algorithm was tested on several instances of the proposed transportation problem. In each run we have measured the number of steps of the algorithm. We have measured separately the number of resolution and labeling steps. In addition we counted the number of actions that were considered to be a part of the partial plan.

Plan length / Steps / States		4	5	6	7	8	9	10
Resolution steps	no	33	39	121	137	218	259	451
	yes	*22*	*25*	*52*	*59*	*117*	*127*	*337*
Labeling steps	no	15	19	44	55	145	163	1179
	yes	*12*	*15*	*18*	*22*	*51*	*62*	*695*
Actions considered	no	7	8	27	28	51	52	71
	yes	*4*	*5*	*10*	*11*	*21*	*21*	*49*

Figure 3: Empirical results for models with and without states

The performed experiments showed that if there is only one goal in the planning problem, the number of steps of the algorithm was almost the same no matter whether the state management was used or not. But note that if the state management is used more variables are necessary be assigned. The same situation appeared when there were more goals in the planning problem but independent on each other (this corresponds to the situation with two or more transporters and the same number of packages to transport).

We have obtained different results for problems with more than one goal which were interacting. This corresponds to the situation when there are fewer available transporters than the number of packages to transport. The state management within the constraint model seems promising in this situation. The results for problems with independent goals are listed in figure 3.

The current stage of the implementation does not allow us to confirm the corresponding improvement in time when state management is used. Our testing implementation has very different costs of the operations. So we cannot make any conclusion about the time improvement at the current stage.

Conclusion and Future Work

We studied the utility of integration of state management into plan-space planning from the constraint programming perspective. We proposed the constraint model for expressing partial plans. The world states are maintained as the actions are sequenced in the resulting plan. We propose special variables and constraints suitable for expressing such world state sequencing.

We have also designed a special solving algorithm for the proposed model. The algorithm dynamically builds the constraint model as the uncertainty is reduced. Our empirical evaluation showed that the state management within the model significantly reduces the number of steps of the algorithm in certain situations. Namely when the planning problem requires satisfying of multiple interacting goals the state management seems to be useful. We cannot say the same about the time improvement at current stage. It is a matter of further research and testing.

We plan to use the proposed state management in the constraint model for an integration of domain specific heuristics into the solving algorithm. The algorithm makes many decisions during solving the planning problem. All these decisions can be more targeted with the aid of the appropriate domain specific heuristics.

Acknowledgement

The research is supported by the Czech Science Foundation under the contract no. 201/04/1102 and by the Grant Agency of Charles University (GAUK) under the contract no. 326/2006/A-INF/MFF. I would like to thank anonymous reviewers for their useful comments.

References

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

Allen, J.; Hendler, J.; and Tate, A. (editors). 1990. *Readings in Planning*. Morgan Kaufmann Publishers.

Barták, R. 2003. *Dynamic Global Constraints in Backtracking Based Environments.* Annals of Operations Research, no. 118, 101-119, Kluwer.

Bernard, D.; Gamble, E.; Rouquette, N.; Smith, B.; Tung, Y.; Muscettola, N.; Dorias, G.; Kanefsky, B.; Kurien, J.; Millar, W.; Nayak, P.; and Rajan, K. 1998. *Remote Agent Experiment. Deep Space 1 Technology Validation Report.* NASA Ames and JPL report.

Blum, A. L., and Furst, M. L. 1997. *Fast planning through planning graph analysis.* Artificial Intelligence 90, 281-300, AAAI Press.

Bonet, B., and Geffner, H. 2001. *Heuristic Search Planner 2.0.* AI Magazine 22(3), 77-80, AAAI Press.

Dechter, R. 2003. *Constraint Processing*, Morgan Kaufmann Publishers.

Dechter, R., and Dechter, A. 1988. *Belief Maintenance in Dynamic Constraint Networks*. In Proceedings AAAI-88, 37-42, AAAI Press.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. *Temporal Constraint Networks*. Artificial Intelligence 49, 61-95.

Fikes, R. E., and Nilsson, N. J. 1971. *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence 2, 189-208.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated Planning: theory and practice.* Morgan Kaufmann Publishers.

Haslum, P., and Geffner, H. 2001. *Heuristic planning with time and resources.* In Proceedings of ECP-2001 4/2001, Springer-Verlag.

Mackworth, A. K. 1977. *Consistency in Networks of Relations.* Artificial Intelligence 8, 99-118.

Nareyek, A.; Freuder, E.C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H. A.; Rintanen, J.; and Tate, A. 2005. *Constraints and AI Planning.* IEEE Intelligent Systems 20(2), 62-72.

Nau, D. S.; Regli, W. C.; and Gupta K. S. 1995. *AI Planning versus Manufacturing Operation Planning: A Case Study.* In Proceedings of IJCAI-95, 1670-1676, Morgan Kaufmann.

Régin, J. C. 1994. *A Filtering Algorithm for Constraints of Difference*. In Proceeding of AAAI-94, 362-367, AAAI Press.

Sacerdoti, E. 1975. *The Nonlinear Nature of Plans.* In Proceedings of IJCAI-75, 206-214.

Schulte, C. 2002. *Programming Constraint Services: High-Level Programming of Standard and New Constraint Services.* LNCS 2302, Springer-Verlag.

Van Beek, P., and Chen, X. 1999. *A Constraint Programming Approach to Planning*. In Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 585-590, AAAI Press.

Vidal, V., and Geffner, H. 2004. *Branching and Pruning: An Optimal Temporal POCL Planner Based on Constraint Programming.* In Proceedings of AAAI-2004, 570-577, AAAI Press.

Interleaving Planning and Scheduling: a Collaborative Approach

Antonio Garrido and **Eva Onaindía**
Dpto. Sistemas Informaticos y Computacion
Universidad Politecnica Valencia (Spain)
{agarridot,onaindia}@dsic.upv.es

Abstract

This paper proposes an approach to interleave planning and scheduling when dealing with real-world problems in a collaborative way. Hence, the paper analyses some challenging points for this collaboration, such as modelling the problem in a joint way, introducing the integrated architecture and, particularly, the definition of planning and scheduling conflicts and the way they are solved when both processes work together. We also include an example to illustrate the interaction of both processes.

Introduction and Motivation

The development of planning and scheduling (P&S) techniques allow to tackle real-world problems that require handling duration of activities, reasoning on resource usage, execution costs, optimisation criteria, etc. in a wide variety of problems (Ghallab, Nau, & Traverso 2004; Smith, Frank, & Jónsson 2000). The aim of planning is to find out **which** actions need to be executed to achieve the problem goals, whereas scheduling aims at finding out **when** these actions need to be executed and with **which** resources, which usually entails a constraint satisfaction process to validate all the problem constraints.

Although it is clear that real-world problems require features of both planning and scheduling, these two processes have been traditionally managed separately and, consequently, this has been the most straightforward approach. However, this approach does not seem sensible: selecting an action in a plan is usually conditioned to several temporal constraints, resource availability and criteria to be optimised. Consequently, planning and scheduling complement each other perfectly, which clearly motivates the challenge in designing flexible models to integrate P&S capabilities. After some attempts to integrate P&S systems (Bartak 2004; Chien *et al.* 2000; Muscettola 1994), the question of how best to design such inter-leaved systems still remains open (Smith, Frank, & Jónsson 2000; Smith & Zimmerman 2004). There are many subtle details for a collaborative system, but the main open points for this collaboration, which also represent the main **contributions** of this paper, are:

- How to model the real problem and specify features of both planning and scheduling. The definition of the basic problem elements is essential to identify the requirements of the integrated system.
- How to define the system architecture to bridge the gap between planning and scheduling. Since both processes do have their own role in the problem solving, we should take this fact into account when designing the system architecture.
- How to identify the most adequate P&S technology for the system, particularly considering that they use similar techniques (graph search, heuristics, constraints management and reasoning, etc.) and the necessities of sharing and exchanging information between the two processes.
- How to solve the different types of P&S conflicts that may arise in a mixed-initiative way, thus analysing the way in which the two processes cooperate, communicate and interact.

The objective of this paper is to shed some light about these points and present an architecture for integrating planning and scheduling as a model that dynamically interleaves a planner and a scheduler.

Modelling Planning and Scheduling Problems. A Brief Description through an Example

A P&S problem involves the execution of a sequence of actions, which must satisfy several constraints (both temporal and on resource availability), in order to achieve some goals, while trying to optimise a metric function. Let us assume a problem inspired by a planetary rovers scenario[1] (see Figure 1) that we will use throughout the rest of the paper. This problem requires that a collection of rovers navigate a planet surface, finding samples, taking images, etc., and communicating them back to a lander. Concurrent use of the rovers must be coordinated with the bottleneck of communication, which must be performed only when the lander is visible from the rover. Further, rovers consume energy in their various activities and they can only be recharged at locations

[1]This problem was introduced in the International Planning Competition 2002 (`http://ipc.icaps-conference.org`) and used since then

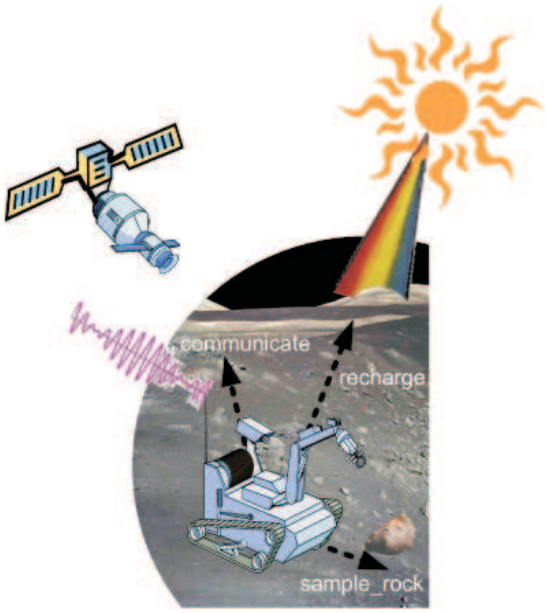


Figure 1: A rovers scenario where rovers have to perform sampling tasks and communicate results back to a lander when visible. The use of rovers must be properly coordinated since tasks require navigation, consume energy and they can only be recharged at particular locations with sun.

that are in the sun (actions to recharge are very limited and need to be planned precisely), which implies to manage energy efficiently. Therefore, trying to separate the navigation and communication part from managing energy consumptions and recharging is nearly impossible. This is a clear example of a real-world problem that makes it necessary to include temporal constraints and resource availability into planning as they modify the structure of the plan; i.e. the plan generation is highly influenced by the scheduling constraints, and separating planning and scheduling turns unlikely. The requirements to model a P&S problem like this rovers scenario include:

- Initial state and problem goals with the information that is true at the beginning of the problem and the facts that need to be achieved, respectively. E.g. the locations with sun, the initial energy of the rovers and their availability, etc. is part of the initial state, whereas the data about samples (rocks, soil and images) to be communicated is part of the goals.
- Actions, with duration, conditions and effects, thus allowing different alternatives to achieve the goals. E.g. the actions allow rovers to navigate, calibrate their cameras, recharge energy, and take images among others.
- Resources available to execute the actions in the plan. In general, this is implicitly included in the actions as conditions and/or effects, and usually even represented as numeric expressions. E.g. the rovers, the cameras, energy, time, etc. are the resources, and actions modify their propositional state (available, calibrated, empty, etc.) and numeric value (decrease energy in 10 units or increase total time of the plan –makespan– in 5 units).
- Metric function, as a multi-criteria function defined by the user that needs to be optimised. The application of this metric allows to find plans where several weighted criteria that play an important role in the plan are considered. E.g. a metric to minimise the plan makespan and the energy used by all the rovers helps assess the quality of different plans to choose the most appropriate.
- Problem constraints (including temporal and non-temporal constraints), with additional constraints the plan needs to satisfy obligatorily (strong constraints) or as a preference (soft constraint). E.g. some constraints such as deadlines in (sub)goals (communication needs to be performed before 30') or constraints in the form of time windows (sun is available from 6' to 150') help better model the requirements of the real problem.

Although there is not a well accepted language to specify real problems of planning and scheduling (at least to the authors' knowledge), from a planning perspective there exists a widely accepted language to define planning domains, which is now called PDDL3 (Gerevini & Long 2005). Despite some limitations in PDDL3 to describe scheduling features such as explicit resources and complex temporal constraints, a high number of features of real-world problems can be currently modelled in PDDL3, which makes it expressive enough to model P&S problems.

Collaborative Planning and Scheduling

The resolution of P&S problems has followed two different perspectives. In one of the research directions, the **temporal planning approach**, the objective is to extend planning to cope with scheduling capabilities, that is augmenting the planning reasoning capabilities in order to handle time and resources (Chen, Hsu, & Wah 2004; Gerevini *et al.* 2004; Ghallab, Nau, & Traverso 2004). In the other research direction, **planning embedded into scheduling**, the solution consists of including planning capabilities into a scheduler (Smith & Zimmerman 2004). In this approach, the starting point is usually a pre-planned set of ordered activities and planning is called each time it is necessary to release, to set up or to make available any problem component. However, the solution does not necessarily go through any of these two approaches since this indistinguishable mixture of processes may increase the complexity of the overall approach, making some problems intractable. Our contribution is to provide a general and flexible model where both planning and scheduling processes have an important role in the problem solving per se. Our two main goals when designing the system architecture are:

- To have two separate but coordinated P&S components, each specialised in solving different problems. First, a planner that initially does not contemplate time and resources, i.e. it only considers the propositional part of the problem in a STRIPS-like style (Fikes & Nilsson 1993). Second, a scheduler that assists the planner to accomplish a plan that accommodates and satisfies the time/resources

requirements. Even though this might sound inefficient for simple problems, it turns out to be extremely effective for solving complex problems with complex constraints.

- To come up with a model where both P&S tasks can be solved by applying existing techniques. There exist many planning and scheduling techniques that, properly combined, can be used for our purpose of creating a cooperative and highly-coupled integrated model of planning and scheduling.

Architecture. Planning and scheduling modules

Figure 2 depicts the structure of the integrated architecture for planning and scheduling, which contains four identifiable elements:

1. The **input data**. The input is the problem model (domain+problem definition in any specification language, e.g. PDDL3) together with an initial plan (either provided by the user as a set of activities or by a planner) used as a basis. Although the system could start from an empty plan, it does not seem sensible to tackle such a complex task from scratch since there are many state-of-the-art planners that generate plans very efficiently. The underlying idea is to use a classical planner, as simple (in terms of expressivity and calculus) and efficient as possible. Note that this does not imply a loss of generality: a plan provided by any planner can be used as an input, and even a plan generated by hand. Also note that this plan may abstract out the scheduling (time+resources) requirements, i.e. the plan does not need to be executable because the objective of the integrated module is precisely to repair a given plan and make it fully executable *w.r.t.* all the scheduling requirements. This increases the opportunities to use efficient planners as a previous step to the integrated module.
2. The **action network** (AN). The input plan is converted into a network that represents the plan with its actions, causal links and (temporal+resource) constraints. Similarly to the activity network used in (Fratini & Cesta 2005), this network represents: i) the shared data structure between the planner and the scheduler; and ii) the shared search space. The AN (see an example in Figure 4 below) follows the philosophy of temporal constraint networks and consumable resource networks (TCN and rCN (Dechter, Meiri, & Pearl 1991; Wallace & Freuder 2005), respectively) to represent the plan, actions and constraints on time and resources. Nodes represent timepoints where actions start/end, while edges, all of them labelled with an interval, represent: i) usage of a resource (time is considered as a resource as well); ii) causal links between timepoints of actions; and iii) temporal constraints between timepoints. The AN also includes information about *obligatory actions* to indicate that these actions must be present in all the plans because they are the only way to support some problem (sub)goals, similarly to the work on landmarks presented in (Hoffmann, Porteous, & Sebastia 2004).
3. The **planning process**. This process updates the AN when solving the planning conflicts and makes actions propositionally executable. This process acts as a plan repairer when it works upon the input plan (adding or deleting actions to make the plan executable), as a replanner when a conflict is detected in the plan or simply as a planner when it deals with goal achievement.
4. The **scheduling process**. This process validates and checks the feasibility of the AN and its constraints. The scheduler performs two important roles in the overall process, one as an autonomous task and another as a cooperative task with the planner. The former involves allocating actions so as to fulfil the temporal constraints and the resource requirements. The latter informs and collaborates with the planner when it detects a scheduling conflict that requires planning abilities to solve it.

The planning and scheduling processes are specially devised to work together in the resolution of the different conflicts that arise during the plan construction (or reconstruction). The proposed architecture brings twofold **advantages**:

- The input data to the integrated model can be a propositionally executable plan, an incomplete plan or an empty plan given by any planner. It is also possible to provide the model with a plan that already satisfies some of the problem constraints. In any case, all this information is transformed into the appropriate elements in the AN. The idea is to provide a flexible input since for some problems it is possible to compute a plan in advance, for some other problems the knowledge engineer or the user can supply some activities (actions) that are known to be necessary in the problem, and for some other complex problems we might not have this type of information or even be very difficult to compute a plan by any standard planner.
- This architecture provides a high flexibility, being possible to plug-in additional specialised modules. The proposed scheduler is thought to work as a TCN by propagating and validating the time and resource constraints feasibility, and providing useful information to help the planner repair the possible conflicts. The same schema can be used for other modules, as for example a module in charge of checking hard and soft constraints among planning states and plan trajectories (Gerevini & Long 2005) or a module specialised in some specific type of constraints. The overall idea is that each of the modules attempts to maintain the consistency of their constraints and collaborate with the planner when conflicts arise in the AN.

Technology. Planning and scheduling techniques

The model for this integration tries to be general enough to use already existing techniques for planning and scheduling. The main goal of the planning process is to get an executable plan without taking into account the resource constraints. This process carries out the typical operations to solve planning conflicts, insert actions to support new goals and so on. Therefore, no special requirements are needed and most current (heuristic) planning and plan repair techniques can be used (Chen, Hsu, & Wah 2004;

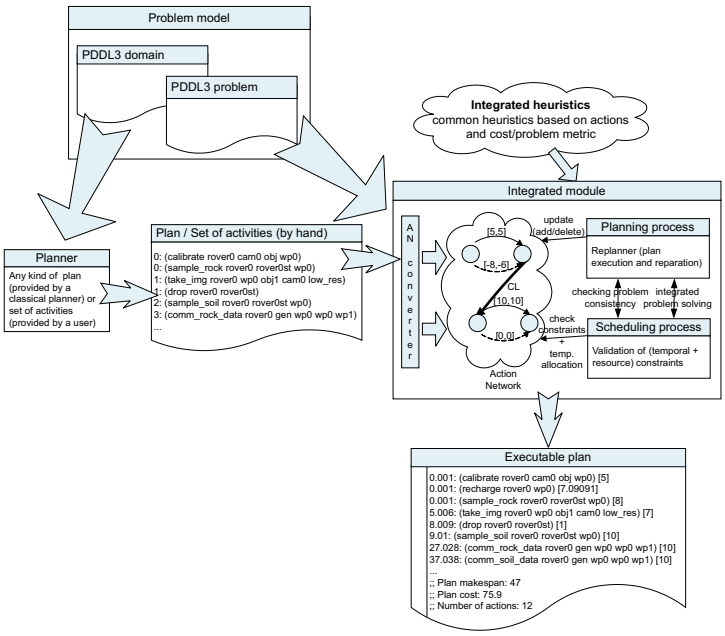


Figure 2: Structure of the collaborative system for planning and scheduling.

Gerevini *et al.* 2004; Ghallab, Nau, & Traverso 2004; van der Krogt & de Weerdt 2005). On the other hand, the scheduling process needs to validate and guarantee the feasibility of the AN. Here different techniques can be used. From the AI point of view, scheduling algorithms for checking the consistency of the constraints involved in the AN (such as TCN (Dechter, Meiri, & Pearl 1991)) are more appropriate than dynamic CSP that try to find a solution to the problem (Tsang 1993). From the OR point of view, there exist many efficient algorithms that can also be used to perform the allocation of the actions in the AN while checking its feasibility (Baptiste, Le Pape, & Nuijten 1995; Winston 1994). In both cases, the scheduler needs to provide information about where the conflicts appear.

As can be seen, the open design of this model represents an important advantage since it allows to use and adequately combine existing techniques and implementations for planning and scheduling to achieve a bi-modular solver. However, such a combination of different techniques makes it difficult to guarantee some properties such as completeness and optimality. Fortunately, when solving complex real-world problems, these two properties are not the most desired properties in our approach.

Interaction. Solving planning and scheduling conflicts

Solving problems that include temporal/resource constraint is a complex task that becomes especially tricky in a collaborative approach for P&S because there exist features that are better managed by these two processes by separate, but there exist others that require a strong stage of interaction and cooperation between both processes. This is particularly more evident when solving P&S conflicts, which needs a deep explanation.

Types of P&S conflicts

We have identified three types of conflicts, which usually arise when generating a plan that must fulfil both P&S features, and must be solved by the processes by separate (type 1 and 2) or by a close collaboration of both of them (type 3). These types are:

- **Type 1**. Planning conflicts solved via planning. These conflicts are the classical ones that are caused by unsupported action preconditions, threats or by mutex relations between the actions in the plan. E.g. if an action in the AN requires to take an image at a particular location while the rovers is in a different point, it becomes necessary to support such a condition by planning new actions. This type of conflict is related to propositional information (pure STRIPS) that is specifically managed by the planning process. Therefore, these conflicts are solved by applying classical planning techniques (flaw repair mechanisms, such as inserting and deleting actions to satisfy unsupported preconditions).
- **Type 2**. Scheduling conflicts solved via scheduling. These conflicts are those caused by oversubscripted resources that must be allocated to competing actions. E.g. if two actions in the AN use the same resource with unitary capacity, which cannot be shared at the same time, it becomes necessary to impose an order of execution between them. This type of conflict is basically related to resource information and it can be implicitly represented in the action as (STRIPS) propositional information or as numeric information. Since these conflicts imply an

assignment problem, they can be solved by scheduling techniques that shift the actions until finding a feasible allocation.

- **Type 3**. Scheduling conflicts solved via planning. Unlike type 2 conflicts, there exist other conflicts that cannot be strictly solved by shifting or re-allocating actions since they require to include a collection of setup actions (subplan) or template that cannot be pre-planned in a domain-independent context (Smith & Zimmerman 2004). E.g. if an action in the AN requires an energy level that exceeds the current value, it becomes necessary to recharge. However, if the action for recharging is conditioned by the result of a collection of actions (i.e. the rover needs to navigate to a particular location), a conflict similar to type 1 appears since new subgoals need to be achieved. This type of conflict is related to resource information and sometimes to propositional information for the setup actions. Solving this conflict is a highly complex process since it first needs scheduling to detect it and second planning to: i) find the place in the plan where repair (subplan) is needed; and ii) modify the original plan (including and/or deleting actions) in order to make the resource available to be used again.

As can be noticed, dealing with P&S features and solving their conflicts is the main task to be done in the collaborative approach. Algorithm 1 shows the scheme for this collaboration, which also represents the functional behaviour of the integrated model when solving the previous types of conflicts. The algorithm works by interleaving the P&S stages in turns and the goal is to revise, both from the planning and scheduling points of view (`rev_PLN` and `rev_SCH`), all actions in the AN to find a conflict-free plan. Selecting one action at a time (step 5), three are the main (planning+scheduling) tasks to perform. One is solve_PLN_conflicts (step 8), responsible for solving type 1 conflicts. Once the current planning conflicts have been solved, the scheduler revises the scheduling conflicts (steps 9–14). First, it searches for a consistent allocation of the actions in `rev_SCH` (steps 11–12) to solve type 2 conflicts. If this allocation fails, the conflict then becomes a type 3 one (solve_SCH_conflicts in step 14) which means it is necessary to modify the AN (insert/delete actions) to solve it.

The key points in Algorithm 1 are the two calls (steps 8 and 14) which correspond to the resolution of type 1 and 3 conflicts. The resolution of these conflicts require planning and scheduling. The overall idea is to **reformulate** a planning conflict as a planning repair problem which is later solved by classical planning techniques. On the other hand, the scheduler provides the planner with advice about the conflicting actions and the resources involved there. Hence, the scheduler helps the planner focus its search upon those planning alternatives that better accomplish the time/resource objectives/constraints. Particularly, the P&S interaction for solving these conflicts is (see Figure 3):

1. solve_PLN_conflicts (type 1 conflicts). In this case, the planner needs to solve a planning conflict in `rev_PLN`. First, the planner uses a heuristic estimator, which is later explained, to find out the best alternatives to repair the conflict. Particularly, the heuristic estimator informs about the different places (states in the AN) where the repair turns out to be feasible, obviously from a heuristic point of view. Starting from the most promising state, the planner tries to solve the conflict by transforming it into its own terms: the planner formulates a repair problem in terms of which subgoals are to be achieved and which ones are to be maintained in the AN. In order to deduce this information, the planner makes use of its own planning tactics to decide which actions need to be inserted into the AN. The interesting point here is that the planner also counts on the information provided by the scheduler (heuristic selector), thus guiding the planner process and helping decide which planning alternative to take on according to scheduling criteria. This way, the planner shows the scheduler the planning decision points (plans $P_1, P_2, \ldots P_5$) and the scheduler determines the best alternative (P_2) from the time/resource perspective. Finally, if the conflict cannot be solved by just inserting new actions, the planner selects another alternative place (state) from the heuristic estimator and repeats the repairing process. The planner tries to solve the conflict on the basis of the current AN, and it will only take the decision of deleting actions from the AN when there are no more alternatives (states) to try out. Deleting actions obviously produces new conflicts since the goals of the deleted actions are not longer supported. Consequently, the planner needs to select a new conflict as a new type 1 conflict, which means starting the process again.
2. solve_SCH_conflicts (type 3 conflicts). In this case, the scheduler needs to solve the scheduling conflict in `rev_SCH`. However, this implies solving a scheduling conflict through planning in a joint way. When the scheduler detects a constraint violation due to an oversubscripted resource $\mathcal{R}$ that cannot be solved by the scheduler per se (a subplan is required), it informs the planner about **when** that resource is needed (at which $time\_point$), **how much** is required ($quantity$) and **who** requires it (action a_i),

1: `rev_PLN` $\leftarrow$ `IS` {`IS`: *fictitious action for the initial state*}
2: `rev_SCH` $\leftarrow \emptyset$
3: **while** $\exists a \in$ `AN` $\mid a \notin$ `rev_SCH` **do**
4: {*Planning part; reasoning on causal links*}
5: $a \leftarrow$ earliest action that can be planned in `AN`
6: `rev_PLN` $\leftarrow$ `rev_PLN` $\cup \{a\}$
7: **if** number_of_conflicts(`rev_PLN`) > 0 **then**
8: solve_PLN_conflicts(`rev_PLN`) {***Type 1** conflicts*}
9: {*Scheduling part; reasoning on time and resources*}
10: **for all** $a_i \in$ `rev_PLN` $\mid a_i \notin$ `rev_SCH` **do**
11: **if** $\exists$ a consistent allocation of a_i in `rev_SCH` **then**
12: `rev_SCH` $\leftarrow$ `rev_SCH` $\cup \{a_i\}$ {***Type 2** conflicts*}
13: **else**
14: solve_SCH_conflicts($a_i, \mathcal{R}, quantity, time\_point, Other\_Constraints$) {***Type 3** conflicts*}

Algorithm 1: General scheme for collaborative planning and scheduling, and the way conflict resolution is invoked

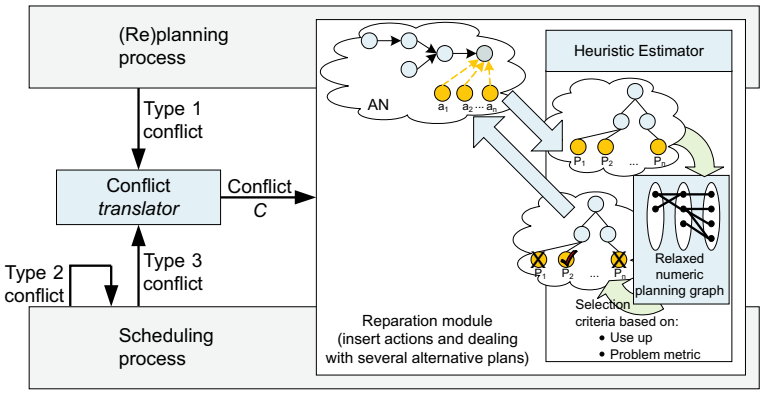


Figure 3: Planning and scheduling interaction for solving type 1 and type 3 conflicts.

thus creating a new subgoal to be supported. Additionally, the scheduler provides the planner helpful information as an extra set of constraints ($Other\_Constraints$) that the planner must hold when solving the problem; for instance, do not use more than n units of resource R_j when solving this conflict. Basically, the planner uses this information and acts similarly as when solving a type 1 conflict, i.e. it uses the heuristic estimator to find the best states in the AN to insert the subplan to solve the conflict that appears with the resource $\mathcal{R}$. Note that there is no guarantee of solving this conflict just by inserting actions. Consequently, in cases of deadline-related conflicts deleting actions becomes the only valid alternative.

Solving conflicts, especially type 1 and 3, require a high component of search and thus the use of heuristic estimations to guide this process appears very convenient.

Heuristic tactics to help solve conflicts Figure 3 shows two heuristic processes that help in the P&S interaction: the **heuristic estimator** in the planner and the **heuristic selector** in the scheduler. The former provides heuristic estimations to discover the best state in the AN from which repairing the conflict (inserting the subplan). These estimations are calculated on a relaxed version of a numeric planning graph, which is similar to the one used in metric-FF (Hoffmann 2003) but with two differences. Firstly, we ignore the action conditions related to the use of numeric resources, which usually denote consumable resources. Secondly, we consider all action effects related to the numeric resources either they imply an increase or decrease in the resource usage. This way we will be able to check the goodness of estimations that consider a global use of resources (consumption/restock). Intuitively, ignoring the numeric conditions may lead to include actions in the planning graph although they are not executable, which would entail very optimistic situations that can negatively affect the estimations. However, even ignoring the numeric conditions, if actions finally turn out to be executable, the estimations will be very close to the real values. On the other hand, considering all type of effects allows to obtain more accurate estimations for most resources since their values will be more informed. In short, these estimations are more oriented to finding out the final consumption of all the resources after executing a set of actions rather than to the realisation of those actions.

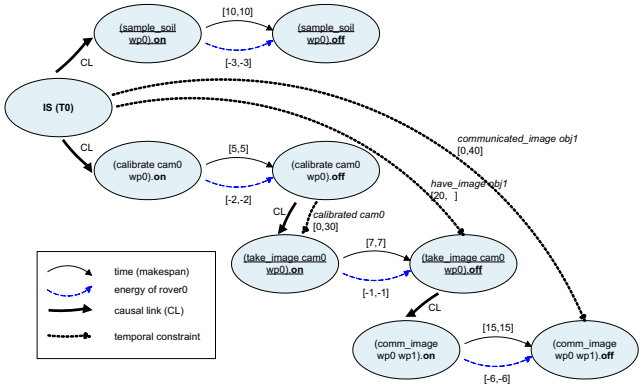
Note that the use of relaxed planning graphs is not only valid for estimating numeric resources but also for any kind of resources that require setup activities such as making available a vehicle fleet, flight crew, etc. In these cases, the estimations will be even more precise since the numeric relaxation has no impact in their values as we will see in the illustration example below.

On the other hand, the heuristic selector is used to choose a plan during the plan repairing process. During this process, the planner will likely find several actions that achieve the same goals, thus forming alternative plans. Instead of doing a complete search, which may be very costly, the heuristic selector informs the planner about which alternative plan to use. The selector makes its decision on the basis of: i) the use of resources; and ii) the multi-criteria problem metric to be optimised. First, it selects plans that use up those resources that require setup activities, thus reducing the number of these activities. For instance, trying to use up the resource energy will reduce the number of actions in the plan (fewer recharge actions will be necessary), the makespan and eventually the cost. Second, it selects plans with the best problem metric which means selecting the plan that best fits the user's preferences. Note, however, that other criteria can also be used, such as number of actions, makespan, etc.

An illustration example Let us take an example of the rovers problem to illustrate the way the planner and scheduler interact when solving conflicts and how the heuristic tactics help in this process. The goal of the problem is to have soil, rock and image data communicated. The initial plan, generated by an automated planner is shown in Figure 4-a, which is subsequently converted into an AN (see Figure 4-b). Moreover, the problem imposes these additional complex constraints: i) actions have different durations and consume energy of `rover0` (e.g. `(sample_soil wp0)` increases time in 10 units and decreases energy in 3 units); ii) the initial energy level of `rover0` is 10; iii) `sun` is available to recharge only in `waypoint1` and `waypoint3`;

```
0: (calibrate camera0 waypoint0)
1: (take_image camera0 waypoint0)
2: (sample_soil waypoint0)
3: (communicate_image waypoint0 waypoint1)
4: (communicate_soil waypoint0 waypoint1)
5: (drop)
6: (sample_rock waypoint0)
7: (communicate_rock waypoint0 waypoint1)
```

(a)



(b)

Figure 4: (a) Initial plan for the illustration example; (b) Fragment of the AN of the initial plan: obligatory actions are underlined, solid and dashed lines represent usage (increase/decrease) of resources in actions (makespan and energy, respectively), thick lines represent causal links, and dotted lines represent temporal constraints.

iv) the subgoal `(have_image objective1)` must not be achieved before time 20; v) the effect `(calibrated camera0)` only persists during 30 units; and vi) the goal `(communicated_image)` has a deadline, so it is due to time 40. In this problem, conflicts arise because of the location of the rover (type 1), its energy level and the makespan, but only the energy and the makespan are managed by the scheduler as numeric resources (type 3 conflicts).

According to Algorithm 1, actions are selected from the AN and revised to check for P&S conflicts. The AN is free of conflicts until revising `(comm_image wp0 wp1)` that introduces a type 3 conflict that the scheduler per se cannot solve because the resource `energy` becomes oversubscripted: `(comm_image wp0 wp1)` requires 6 units of energy, but only 4 (10-3-2-1) are available. Then, the scheduler executes solve_SCH_conflicts((`comm_image wp0 wp1`), (`energy rover0`), 2 (6-4), (`comm_image wp0 wp1`).on, {($\sum$(`energy rover0`) $\leq$ `10`), (`makespan` $< \infty$)}). Hence, the scheduler informs the planner about all the information about the conflict: action, resource, quantity required and timepoint. Moreover, it also informs about the *other constraints* to be satisfied that include: i) consumption of `energy rover0` cannot now exceed 10 (since this is the initial value); and ii) consumption of `makespan` is not really constrained.

The heuristic estimator calculates the four states where

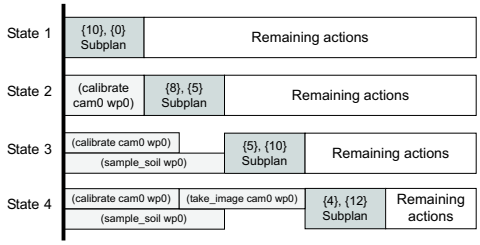


Figure 5: The four states from where the planner can start the subplan to recharge, after executing 0,1,2,3 actions, respectively, of the AN. Each subplan also contains the initial value of the resources `energy` and `makespan`, respectively. Hence, the initial values for the subplan of the state 3 are {5},{10} that coincide with the consumption of both resources of the parallel execution of actions `calibrate` and `sample_soil`.

Table 1: Estimation for the consumption of resources to execute actions to recharge energy inserting the subplan in four different states (values for `wp1` and `wp3` are the same).

	`energy rover0`	`makespan`
State 1	**10-8=2**	**0+5=5**
State 2	**8-8=0**	**5+5=10**
State 3	5-8=-3	10+5=15
State 4	4-8=-4	12+5=17

the subplan to solve this conflict can be inserted, as depicted in Figure 5. The only two actions that increase the energy level are `(recharge wp1)` and `(recharge wp3)` (in this problem they have the same estimations because both waypoints are interchangeable), so the planner has to include a subplan (setup activities) to navigate there, which consumes 8 units of `energy` and 5 of `makespan`. According to the scheduling constraints on the `energy` and `makespan`, the subplan can only be placed in states 1 or 2 (see Table 1). Clearly, state 3 is not a valid choice because the initial `energy` level for the subplan is 5 and it uses 8 units to solve the conflict. The same happens in state 4 where there is a lack of 4 units of `energy` to achieve the subplan. In order to decide between state 1 or 2, the estimator uses criteria based on the use-up of resources and problem metric (as in the heuristic selector). In this case, the state 2 is chosen since the fact of totally using up the resource `energy` allows to minimise the number of recharges and improve the quality of the final plan. Then, the planner will generate the subplan, which consists of actions {(`navigate wp0 wp1`) $\rightarrow$ (`recharge wp1`)} (or analogously in `wp3`), updating the actions in the AN and resuming the Algorithm 1. This shows how the P&S interaction, together with the use of heuristic tactics, allow to face conflict resolution in a collaborative way.

Conclusions through Related Work

In the last decade, combining AI planning and scheduling has posed a lot of interesting and challenging questions, be-

ing considered a hot topic of research even from the OR point of view (Baptiste, Le Pape, & Nuijten 1995; Bartak 2004; Fratini & Cesta 2005; Ghallab, Nau, & Traverso 2004; Smith, Frank, & Jónsson 2000). The combination of P&S techniques varies from loosely-coupled integration (Pecora & Cesta 2005) to highly-coupled integration with a constant interaction of P&S modules (Chien *et al.* 2000; Muscettola 1994). The way to share information between planning and scheduling, as the level of synergy between both solving processes (Pecora & Cesta 2003), becomes a key aspect in both types of integration, being particularly more difficult to achieve in the latter. There have been some successful attempts in such a highly-coupled integration, such as HSTS (Muscettola 1994) or Aspen (Chien *et al.* 2000). These ad-hoc systems exhibit a good performance in the domains they were designed for, but they are limited in more general domains. Therefore, the challenge of designing a general model of planning and scheduling still remains open.

This paper represents a step ahead towards a collaborative approach for planning and scheduling which intends to be general and flexible enough to tackle any kind of P&S problems, with a strong or weak component of time/resource management. Our model is based on the common idea of dynamically interleaving planning and scheduling, both playing a similar role. However, we bring two novelties with respect to other approaches: i) none of the two processes (planner and scheduler) is embedded into the other but both work together in the problem resolution; and ii) the proposed model can be implemented using existing technologies of planning and scheduling. We have not focused on building a complete and optimal approach for planning and scheduling, but on designing the interaction between both processes, which we consider a key point for the success of the collaboration. This implies making decisions on **when** to communicate, **what** to communicate and **how** the processes will tackle the task when they have all the information at hand. In particular, we have shown how the planner and scheduler interact for early detection and solution of conflicts, more specifically for the type 3 conflicts which require capabilities of both processes to be solved. As part of our immediate research, we are focusing our work on: i) how to improve the P&S interaction; and ii) the implementation of the ad-hoc mechanism for providing feedback from the scheduler to the planner and the evaluation of its effectiveness.

Acknowledgements

This work has been partially supported by the Spanish government project MCyT TIN2005-08945-C06-06 (FEDER) and by the Valencian government project GV06/096.

References

Baptiste, P.; Le Pape, C.; and Nuijten, W. 1995. Incorporating efficient operations research algorithms in constraint-based scheduling. In *Proc. 1st Int. Joint Workshop on Artificial Intelligence and Operations Research*.

Bartak, R. 2004. Integrating planning into production scheduling: a formal view. In *Proc. of ICAPS-2004 Workshop on Integrating Planning Into Scheduling*, 1–8.

Chen, Y.; Hsu, C.; and Wah, B. 2004. SGPlan: Subgoal partitioning and resolution in planning. In *Proc. ICAPS-2004 – International Planning Competition*, 30–32.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN - automating space mission operations using automated planning and scheduling. In *Proc. SpaceOps 2000*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Fikes, R., and Nilsson, N. 1993. STRIPS, a retrospective. *Artificial Intelligence* 59(1-2):227–232.

Fratini, S., and Cesta, A. 2005. The integration of planning into scheduling with OMP. In *Proc. Workshop on Integrating Planning into Scheduling (WIPIS/AAAI-05)*.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. Technical report, University of Brescia, Italy.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. Planning in PDDL2.2 domains with LPG-TD. In *Proc. ICAPS-2004 – International Planning Competition*, 33–34.

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

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

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

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. San Mateo, CA: Morgan Kaufmann. 169–212.

Pecora, F., and Cesta, A. 2003. The role of different solvers in planning and scheduling integration. In *Proc. Congress of the Italian Association for Artificial Intelligence (AI*IA 2003)*, 362–374.

Pecora, F., and Cesta, A. 2005. Evaluating plans through restrictiveness and resource strength. In *Proc. Workshop on Integrating Planning into Scheduling (AAAI-2005)*.

Smith, S., and Zimmerman, T. 2004. Planning tactics within scheduling problems. In *Proc. ICAPS-2004 Workshop on Integrating Planning Into Scheduling*, 83–90.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15(1):47–83.

Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proc. ICAPS-2005*, 161–170.

Wallace, R., and Freuder, E. 2005. Supporting dispatchability in schedules with consumable resources. *Journal of Scheduling* 8(1):7–23.

Winston, W. 1994. *Operations Research: Applications and Algorithms*.

Scheduling with uncertain durations: generating β-robust schedules using constraint programming

Christine Wei Wu and **Kenneth N. Brown**
Cork Constraint Computation Center,
Department of Computer Science,
University College Cork, Ireland
{cww1, k.brown}@cs.ucc.ie

J. Christopher Beck
Toronto Intelligent Decision Engineering
Laboratory, Dept. of Mechanical and
Industrial Engineering,University of Toronto,
Canada. jcb@mie.utoronto.ca

Abstract

Many real-world scheduling problems are subject to change, and scheduling solutions should be robust to those changes. We consider a single-machine scheduling problem where the processing time of each activity is characterized by a normally-distributed random variable, and we attempt to minimize flowtime. We develop an initial constraint model for generating the β-robust schedule - the schedule that has highest probability of producing a flowtime less than a stated bound. Experiments with this initial model show that a constraint-based approach is feasible, but that better propagation methods will be required.

Introduction

Most scheduling research considers problems that are static and certain – all the activities and their durations are known in advance and do not change as the solution is being executed. However, many real-world scheduling problems are subject to change: new jobs arrive, resources fail, or tasks take longer than expected. If these changes are significant, then optimal solutions to the original problem may turn out to be poor in practice. For this reason, it may be better to generate solutions that are robust to the likely changes. A β-*robust* schedule (Daniels & Carrillo 1997) is one that has maximum probability of achieving a given performance level (e.g. total flowtime less than a threshold). Alternatively, we may want to find the best performance that a solution will deliver with a given confidence level.

Constraint-based methods have proven to be very effective in a wide range of industrial scheduling problems. The advantage comes from the flexibility of the modeling language, and the ability of the solvers to deliver effective performance despite the presence of a wide range of different constraints and objectives. Again, though, most constraint based research assumes static and certain problems. In this paper, we consider how to model β-robustness in a constraint modelling language, and we show how to search for β-robust schedules.

In particular, we consider single machine problems, where the processing time of each task is uncertain, but can be characterized by a normally-distributed random variable. We consider flowtime (the amount of time the tasks remain in the system) as the main criterion. The simplest approach would focus on optimizing the expected total flowtime. However, this ignores the variance in the task durations, which may be significant. For any given schedule, we will measure the probability of the total flowtime being less than a target level. We will then generate (i) a schedule which maximizes the probability, or (ii) a schedule which optimizes the target level that can be achieved with a given probability.

The paper is structured as follows: first, we briefly review techniques for scheduling under uncertainty; we then consider flowtime as a performance measure for schedules with uncertain task durations; we give a formal definition of β-robustness; we present our initial constraint models for the β-robust scheduling problem; and finally we report on some experiments with the model.

Background

A number of approaches have been proposed to handle uncertain scheduling problems. Redundancy-based Scheduling generates schedules with temporal slack so that unexpected events during execution can be handled by using that reserved slack (Davenport, Gefflot, & Beck 2001; Gao 1995). Contingent scheduling anticipates likely disruptive events and generates multiple schedules which optimally respond to the anticipated events (Drummond, Bresina, & Swanson 1994; Fowler & Brown 2003). Probabilistic scheduling uses probabilities over possible events, and searches for schedules which optimize the expected value of some performance measure (Daniels & Carrillo 1997; Walsh 2002; Beck & Wilson 2004; 2005). A number of approaches use sampling and scenarios, in order to produce robust decisions (Bent & Hentenryck 2004; Beck & Wilson 2004).

In particular, Daniels and Carrillo (Daniels & Carrillo 1997) introduced the concept of the β-robust schedule for a single machine scheduling problem with processing time uncertainty. They solved the problem by a branch-and-bound method with dominance rules, and heuristics for branch selection. The total flowtime was used to measure the performance of solutions, which will be explained in the next section.

The flowtime of a schedule

In a single machine scheduling problem, in which each job consists of a single task, a machine can only process one job at a time, and a job cannot be interrupted once started, a solution is a sequence of the jobs, and we assume the jobs are executed in sequence with no delay between them. Suppose we have a sequence $J_1, J_2, \ldots J_n$. Each job J_i has an arrival time A_i (its earliest possible start time), a start time ST_i, a duration d_i, and an end time E_i. We assume that each job is available for processing at time 0 (i.e. $A_i = 0$).

We note the following simple relations: $E_i = ST_i + d_i$, $ST_1 = 0$, $ST_i = E_{i-1}$, and hence $E_i = \sum_{j=1}^{i} d_j$.

The *flowtime* is the total time the jobs are in the system:

$$TFT = \sum_{i=1}^{n} (E_i - A_i)$$

Because we assume $A_i = 0$, we can rewrite the equation for total flowtime as follows:

$$\begin{aligned} TFT &= \sum_{i=1}^{n} E_i && (1) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{i} d_j && (2) \\ &= \sum_{i=1}^{n} (n+1-i) * d_i && (3) \end{aligned}$$

We now assume that each job J_i's duration is an independent normally distributed random variable $d_i \sim N(\mu_i, \sigma_i^2)$. We assume that the jobs will still be executed in the given sequence, regardless of the actual values of the durations.

We note that for any two independent random variables $X \sim N(\mu_x, \sigma_x^2)$ and $Y \sim N(\mu_y, \sigma_y^2)$, and two constants a and b, the sum $aX + bY$ is also a normally distributed random variable, such that $aX+bY \sim N(a\mu_x+b\mu_y, a^2\sigma_x^2+ b^2\sigma_y^2)$.

Since the activity durations are independent normally distributed random variables, and flowtime is a linear combination of durations, then for any particular sequence of jobs, the flowtime is also a normal random variable. From (3):

$$TFT \sim N(\sum_{i=1}^{n} (n-i+1)\mu_i, \sum_{i=1}^{n} (n-i+1)^2 \sigma_i^2)$$

β-robust schedules

For scheduling problems with uncertainty, we must decide the criteria by which the solutions will be judged. The simplest criterion is the expected flowtime (or the average actual flowtime over a number of runs). In this case, the scheduler only needs to consider the expected parameters of the individual jobs. However, in real settings, some form of service level may be more important – what level of confidence can a customer or manager have in predicted performance levels? Rather than gambling on the expected performance, it may be more useful to give a lower limit on the performance, and to state the confidence in being able to achieve that level. In this case, it is not enough to know the expected values of the job parameters – the scheduler must also reason about the variance of those parameters in order to determine the variance of the schedule as a whole.

For example, consider the simple problem consisting of three jobs $\{x, y, z\}$, with uncertain durations $\{d_x \sim N(9,2), d_y \sim N(5,1), d_z \sim N(8,7)\}$. The sequence $s_e = \langle y, z, x \rangle$ has a flowtime which is distributed as $N(40, 39)$. 40 is, in fact, the smallest expected flowtime possible for this problem. An alternative sequence, $s_\beta = \langle y, x, z \rangle$, has flowtime $\sim N(41, 24)$, and thus has a higher expected flowtime. However, suppose we now introduce a desired maximum flowtime of (for example) 51: the scheduler will incur a penalty if the actual schedule has a flow time greater than 51. Sequence s_e has a probability of 0.04 of producing a flowtime greater than 51, while s_β has a probability of just 0.02 of delivering a flowtime greater than 51, and thus s_β is likely to be less expensive. s_β is the *β-robust* (Daniels & Carrillo 1997) schedule for the maximum flowtime of 51 - that is, it has the highest probability of delivering a flowtime no greater than 51. In addition, for the confidence level of 0.98, s_β also delivers the minimal flowtime limit (51).

Definition 1. *For the single machine scheduling problem with n jobs, with normally-distributed uncertain durations, and with a flowtime limit S, the β-robust scheduling problem is to find the sequence, s, which maximizes the probability of the flowtime being less than S. That is, find the s that maximizes $Prob(flowtime(s) \leq S)$ (Daniels & Carrillo 1997).*

First, we show how to compute $Prob(\ flowtime \leq S\ )$ for an arbitrary sequence of the n jobs. Since the random variables in the problem are normally distributed, we can use the formula below to compute the probability of $flowtime \leq S$, where μ is the mean flowtime of the sequence, and σ^2 is its variance:

$$\phi(x \leq X) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{X} e^{\frac{-(x-\mu)^2}{2\sigma^2}} dx$$

An arbitrary normal distribution can be converted to a standard normal distribution by changing variables to $z = (x - \mu)/\sigma$, so the normal distribution function becomes:

$$\begin{aligned} \phi(x \leq X) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{\frac{-t^2}{2}} dt \\ &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{0} e^{\frac{-t^2}{2}} dt + \phi(z) \\ &= \frac{1}{2} + \phi(z) \end{aligned}$$

where

$$\phi(z) = \frac{1}{\sqrt{2\pi}} \int_{0}^{z} e^{\frac{-t^2}{2}} dt.$$

Hence, the probability of $flowtime \leq S$ can be computed by

$$Prob(\ flowtime \leq S\ ) = \frac{1}{2} + \phi(z) \qquad (4)$$

Figure 1: the Constraint Model.

Variables:

Job positions: $Pos_1, ..., Pos_n$
Job mean flowtime contributions:
$meanFTContrib_1, ..., meanFTContrib_n$
Job variance flowtime contributions:
$varFTContrib_1, ..., varFTContrib_n$

Constraints:

`allDifferent`(Job positions)

$meanFTContrib_i = (n - Pos_i + 1)\mu_i$
$varFTContrib_i = (n - Pos_i + 1)^2\sigma_j^2$
$mean(flowtime) = \sum_{i=1}^{n} meanFTContrib_i$
$var(flowtime) = \sum_{i=1}^{n} varFTContrib_i$

Dominance constraints:
for $0 \leq i < j < n$,
$\mu_i \leq \mu_j$ and $\sigma_i^2 \leq \sigma_j^2 \Rightarrow Pos_i < Pos_j$
$\mu_i \geq \mu_j$ and $\sigma_i^2 > \sigma_j^2 \Rightarrow Pos_i > Pos_j$
$\mu_i > \mu_j$ and $\sigma_i^2 = \sigma_j^2 \Rightarrow Pos_i > Pos_j$

$objective = max(z) = max(\ \frac{S - mean(flowtime)}{\sqrt{var(flowtime)}}\ )$

where $S \geq 0$ and

$$z = \frac{S - mean(flowtime)}{\sqrt{var(flowtime)}}. \quad (5)$$

For each possible schedule, we can compute the mean and variance of the flowtime by $mean(flowtime) = \sum_{i=1}^{n}(n - i + 1)\mu_i$ and $var(flowtime) = \sum_{i=1}^{n}(n - i + 1)^2\sigma_i^2$ as in equation (3). Then, $\phi(z)$ can be obtained by checking z in the standard normal distribution table (Z-table).

Alternatively, there is a simple approximation of $\phi(z)$ which is good to two decimal places (Weisstein 2006), given by

$$\phi(z) \approx \varphi(z) \begin{cases} 0.1z(4.4 - z) & (0 \leq z \leq 2.2) \\ 0.49 & (2.2 < z < 2.6) \\ 0.50 & (z \geq 2.6) \end{cases} \quad (6)$$

Theorem 1. *$\varphi(z)$ increases on $[0, +\infty)$. For proof, see appendix.*

The β-robust schedule is one of those alternative sequences of the jobs, such that it has the maximum probability of $flowtime \leq S$. To find a β-robust schedule, we need to have an objective function to maximize the probability. We use the approximation of $\phi(z)$ to compute the probability, because it simplifies the calculation. If $\phi(z)$ is increasing on $[0, +\infty)$, maximizing the probability of $flowtime \leq S$ is the same as maximizing z.

$$\begin{aligned} objective &= max(\ probability\ (flowtime \leq S)\ ) \\ &= max(\ \frac{1}{2} + \phi(z)\ ) \\ &= \frac{1}{2} + max(\ \phi(z)\ ) \\ &\approx \frac{1}{2} + max(\ \varphi(z)\ ) \\ &= \frac{1}{2} + \varphi(\ max(z)\ ). \end{aligned}$$

With above analysis and calculations, we are ready to introduce our constraint models for the β-robust scheduling problem.

Constraint models

We first consider the original β-robust scheduling problem described in (Daniels & Carrillo 1997) as a CSP, and then we propose a variable ordering heuristic.

The Constraint Model is shown in Figure 1. We assume a set $\{J_1, J_2, \ldots J_n\}$ of jobs, each with a normally-distributed random variable duration $D_i \sim N(\mu_i, \sigma_i^2)$. Differing from the previous sections, we now do not assume that the jobs are scheduled in the given sequence. With each job J_i, we associate a position variable, Pos_i, with domain $\{1, 2, \ldots, n\}$. The position variable Pos_i represents the position of J_i in the sequence: for instance, $Pos_2 = 3$ states that J_2 is scheduled to be the third job to start on the machine. Besides position variables, we also introduce additional variables for computing flowtime mean and variance and then the probability. The formula (3) indicates that flow time can be viewed as the sum of the contributions from all jobs. We define flowtime contribution of J_i as $FTContrib_i = (n - Pos_i + 1)D_i$. $meanFTContrib_i$ and $varFTContrib_i$ are the mean and variance of the flowtime contributions from J_i. The former has an integer value in $[\mu_i, n\mu_i]$, and the latter has a value in $[\sigma_i^2, n^2\sigma_i^2]$. The goal is to sequence those jobs, i.e. assign a distinct value to each Pos_i, such that the likelihood of the sequence (schedule) to achieve a fixed system performance level S is optimized, i.e. $Max(\ Probability(\ X \leq S\ )\ )$, where X is flowtime. Alternatively, we can look for a schedule with the optimal system performance S for a fixed probability, i.e. $Min(\ S\ )$ such that $Probability(\ X \leq S\ ) \geq C$, where C is the fixed probability.

We show how to achieve the first goal, i.e. optimizing the probability for a fixed performance. Firstly, we have a permutation constraint that ensures each job takes a different position in the sequence. This can be implemented as a global all-different constraint on all the Pos_i. Also if we consider the flowtime as a sum of contributions from each

job, from formula (3), we have

$$\begin{aligned} mean(flowtime) &= \sum_{i=1}^{n} meanFTContrib_i \\ &= \sum_{i=1}^{n}(n - Pos_i + 1)\mu_i, \\ var(flowtime) &= \sum_{i=1}^{n} varFTContrib_i \\ &= \sum_{i=1}^{n}(n - Pos_i + 1)^2\sigma_i^2. \end{aligned}$$

With those additional variables, we can use formula (4), (5) and (6) to compute the probability of a schedule's actual flowtime being less than S.

We are also able to impose some dominance constraints as in figure 1, using the properties of the β-robust schedule.

Theorem 2. *In a β-robust schedule, if job i with $D_i \sim N(\mu_i, \sigma_i^2)$ precedes job j with $D_j \sim N(\mu_j, \sigma_j^2)$, then either the mean duration of job i, μ_i, is no greater than the mean duration of job j, μ_j, or the duration variance of job i, σ_i^2, is no greater than the duration variance of job j, σ_j^2, that is $\mu_i \leq \mu_j$ or $\sigma_i^2 \leq \sigma_j^2$. (see Appendix for the proof)*

With this property, we post further constraints: for job i and job j ($0 \leq i < j < n$), if $\mu_i \leq \mu_j$ and $\sigma_i^2 \leq \sigma_j^2$, then $Pos_i < Pos_j$; if $\mu_i \geq \mu_j$ and $\sigma_i^2 > \sigma_j^2$ then $Pos_i > Pos_j$; if $\mu_i > \mu_j$ and $\sigma_i^2 = \sigma_j^2$ then $Pos_i > Pos_j$. Note that for the jobs have the same duration mean and the same duration variance, we take the lexicographical order on their indexes, i.e. if $\mu_i = \mu_j$ and $\sigma_i^2 = \sigma_j^2$ and $i < j$, then $Pos_i < Pos_j$.

So far, we have presented our constraint model to achieve the first objective. Our second objective is to minimize system performance S such that there exists a schedule that can achieve S with a fixed probability. That is $Min(\,S\,)$ such that $Probability(\,X \leq S\,) \geq C$, where C is the fixed probability. Using the same constraint model, we can get z value from formula (4) and (6)

$$z = \varphi^{-1}(C - \frac{1}{2}).$$

Then, from formula (5), we have a new objective function

$$min(S) = min(\,z*\sqrt{var(flowtime)}+mean(flowtime)\,).$$

Besides those constraints we discussed above, we also implement a variable ordering heuristic to guide search. From formula (5), we can see that the β-robust schedule has the optimized combination of $mean(flowtime)$ and $var(flowtime)$. In order to find the β-robust schedule more quickly, we prefer to first schedule a job i, which has shorter mean processing time μ_i and smaller variance σ_i^2. We use a family of variable ordering heuristics, ordering the jobs by increasing $\mu_i + q * \sigma_i^2$, selecting a value for q based on the problem characteristics. For the first objective (maximizing the probability), we start by finding the SEPT (shortest expected processing time) schedule; we then compute the probability P of it having a flowtime less than S; and from P we select a value for q from a lookup table based on previous experiments with other problems. For the second objective (minimizing the flowtime target achievable by a given probability), we base q on the probability. In both cases, for higher probabilities, we expect the variance to be more significant, and so we choose higher values of q which give increasing weight to the duration variance in the variable ordering. Example values for q are 0.3,0.6, and 1.0 for probabilities of 0.85, 0.95 and 0.99. Note that this variable ordering heuristic does not improve the total solving speed (i.e. the time of finding the schedule and proving it is the optimal), but does shorten the time to find the optimal solution.

Discussion and Experimental results

We implemented the β-robust scheduling problem as a constraint satisfaction problem using ILOG Scheduler and Solver 6.0. Our first aim is to verify our initial constraint model, and so we expect to see the same pattern of results as obtained by (Daniels & Carrillo 1997). Secondly, we want to determine whether or not a constraint model is feasible for such problems, and so we hope to see runtimes of a similar order of magnitude. If we succeed in both aims, we will then investigate more sophisticated constraint models.

We consider problems with either 10 or 15 jobs, using the same experimental setup as (Daniels & Carrillo 1997). The mean processing time for each job i is randomly drawn from a uniform distribution of integers on the interval $\mu_i \in [10, 50\delta_1]$; the processing time variance of job i is then randomly drawn from a integer interval $\sigma_i^2 \in [0, \frac{1}{9}\mu_i^2\delta_2]$. The parameter δ_1 and δ_2 control the variability in the average processing times in the test problems, which can both take any value of 0.4, 0.7 or 1.0. We are interested in schedules that yield acceptable performance with probability approximately 0.85, 0.95 or 0.99. Ten instances are generated randomly for each combination of number of jobs, δ_1, δ_2, and probability level, resulting in a total 540 test problems.

Table 1 contains the results for our constraint methods and the corresponding figures taking directly from Daniels and Carrillo (Daniels & Carrillo 1997). The CPU is the computation time for finding and proving the β-robust schedule. Since Daniels and Carrillo performed the experiment on a 486 personal computer, our CPU time is not directly comparable with theirs but should give us an indication of whether or not a constraint-based approach is feasible. Table 1 also shows the differences (in average and in maximum deviation) between the mean processing time of the β-robust schedule and the shortest expected processing time (SEPT).

The results in table 1 show that we do have a similar pattern in term of the mean flowtime of the β-robust schedule compared to the SEPT schedule. In addition, our CPU time is comparable for the smaller problems, but is poorer for the larger problems. This indicates that a constraint-based approach may be feasible, but that a more sophisticated model with better propagation will be required. We set up a further experiment to determine the effort require to prove that the solution is optimal. For each probability level, we experimented with 90 problems generated the same way as table

Table 1: Computational performance of β-robust solution procedure.

		Constraint model			Branch-and-bound		
total jobs	prob. level	CPU (sec.)	Avg. abv. SEPT (%)	Max. abv. SEPT (%)	CPU (sec.)	Avg. abv. SEPT (%)	Max. abv. SEPT (%)
10	0.85	0.1	0.1	0.4	0.2	0.1	0.8
	0.95	0.1	0.3	1.7	0.2	0.3	1.9
	0.99	0.1	0.5	1.9	0.3	0.6	2.5
15	0.85	2.3	0.1	0.3	1.0	0.1	0.5
	0.95	2.4	0.2	0.7	1.7	0.2	1.0
	0.99	3.0	0.3	1.5	2.1	0.4	1.9

Table 2: The comparison of finding the optimal result and proving that is optimal.

		Time(s)		Time(%)	
total jobs	prob. level	search	prove	search	prove
15	0.85	0.06	2.28	2.63	97.37
	0.95	0.35	2.50	14.04	85.96
	0.99	1.03	3.84	26.82	73.18

1. We also used the variable ordering heuristic as described in constraint models section. The time of finding the optimal solution and the time to prove it is optimal were recorded in each case. Table 2 shows that it takes little time to find the best solution but usually a long time to prove if it is the β-robust schedule. We believe that a problem is hard for our model if it has many jobs with similar duration mean and variance. The program is able to do little propagation, and thus spends a lot of time trying different permutations of the jobs for no benefit.

With the general model, we can also give the minimum system performance S for a problem, so that the jobs in the problem can be scheduled to achieve the minimized S with a desired probability level. Table 3 shows the time to find the minimum S for 15 jobs problems and the corresponding mean flowtime over the SEPT.

Daniels and Carrillo compared the mean flowtime of the β-robust schedule and the optimal expected flowtime. That indicates how much worse the β-robust schedule is compared to the SEPT schedule. It seems more reasonable to compare the probabilities of achieving the system performance of the β-robust schedule with the SEPT schedule. That is

$$\begin{aligned} & [probability(\ flowtime_{\beta} \leq S\ ) \\ - \quad & probability(\ flowtime_{SEPT} \leq S\ )] \\ \div \quad & probability(\ flowtime_{SEPT} \leq S\ ) \end{aligned}$$

which indicates the benefit of using the β-robust schedule. In fact, when some jobs have the same mean, there are multiple possible schedules with shortest expected processing time, but each will have different robustness levels. Daniels and Carrillo did not report the detail of how they generated the SEPT schedule. In our experiments, we used the existing variable ordering, breaking ties lexicographically.

Table 3: Computational performance of finding the minimum system performance of required probability level (15 jobs).

		Constraint model		
number of jobs	probability level	CPU (sec.)	Avg. abv. SEPT(%)	Max. abv. SEPT(%)
15	0.85	2.63	0.04	0.24
	0.95	3.38	0.16	0.87
	0.99	2.70	0.28	1.11

Future work

We are currently working on the *dual* model of the original (primal) model, and a third model which channels between the other two (Hnich, Smith, & Walsh 2004). We believe using the combined model will help us to improve the solving speed. We also plan to investigate better bounds for pruning branches at the top of the search tree, better heuristics to guide the search, and the construction of a global constraint for achieving β-robustness. We are also conducting an investigation into the characteristics of the problems which make some of them much harder to solve than others. Finally, we plan to extend this work to consider problems with multiple machines and with non-zero arrival times, for which the probability calculations reported here will not apply.

Conclusion

In this paper, we presented a general constraint model for the β-robust scheduling problem, which allows us to produce schedules which are robust to uncertainty in the durations of tasks. With flowtime as the performance measure, we can optimize the probability and find a most promising schedule to satisfy the system performance requirement; or we can optimize the performance level for a fixed probability. Our initial model demonstrates that a constraint-based approach is feasible for this problem, but that more more sophisticated models are required for good performance.

Appendix

Theorem 1. *$\varphi(z)$ increases on $[0, +\infty)$.*

Proof. For all $b > a > 2.2$, it is trivial to see $\varphi(b) \geq \varphi(a)$. For all $0 \leq a < b \leq 2.2$,

$$\begin{aligned} \varphi(b) - \varphi(a) &= 0.1\,b(\,4.4 - b\,) - 0.1\,a(\,4.4 - a\,) \\ &= 0.1(\,4.4b - b^2 - 4.4a + a^2\,) \\ &= 0.1[\,4.4(\,b - a\,) + (\,a + b\,)(\,a - b\,)\,] \\ &= 0.1(\,b - a\,)(\,4.4 - a - b\,). \end{aligned}$$

Since $b > a$,

$$\Rightarrow (b-a) > 0$$

and $0 \leq a, b \leq 2.2$

$$\Rightarrow 0 \leq a+b \leq 4.4$$
$$\Rightarrow 4.4 - a - b \geq 0.$$

Hence,

$$\varphi(b) - \varphi(a) \geq 0$$

for all $0 \leq a < b \leq 2.2$, i.e. $\varphi(z)$ increases on $[0, +\infty)$. □

Theorem 2. *In a β-robust schedule, if job i with $D_i \sim N(\mu_i, \sigma_i^2)$ precedes job j with $D_j \sim N(\mu_j, \sigma_j^2)$, then either the mean duration of job i, μ_i, is no greater than the mean duration of job j, μ_j, or the duration variance of job i, σ_i^2, is no greater than the duration variance of job j, σ_j^2, that is $\mu_i \leq \mu_j$ or $\sigma_i^2 \leq \sigma_j^2$.*

Proof. (by contradiction)
Assume X is a β-robust schedule of n jobs, and there exist two jobs i and j in X such that i precedes j, $\mu_i > \mu_j$ and $\sigma_i^2 > \sigma_j^2$.

Suppose i's position in X is a ($1 \leq a \leq n$) and j's position is b ($1 \leq b \leq n$).
Since i precedes j, we have:

$$1 \leq a < b \leq n.$$

Also, the expected flowtime contribution from i, $meanFTContrib_i$, is $(n - a + 1)\mu_i$ and the expected flowtime contribution from j, $meanFTContrib_j$, is $(n - b + 1)\mu_j$. Now, we consider only swap i and j but keep other jobs' positions unchanged in X to get schedule X'. In X', i's position is b and j's position is a. Then,

$$meanFTContrib'_i = (n - b + 1)\mu_i$$

and

$$meanFTContrib'_j = (n - a + 1)\mu_j.$$

The difference between X and X' in term of the expected total flowtime is:

$$\begin{aligned}
& mean(flowtime) - mean(flowtime)' \\
= \; & meanFTContrib_i + meanFTContrib_j \\
& + meanFTContrib_{rest} - (\, meanFTContrib'_i \\
& + meanFTContrib'_j + meanFTContrib_{rest}\,) \\
= \; & meanFTContrib_i + meanFTContrib_j \\
& - meanFTContrib'_i - meanFTContrib'_j \\
= \; & (n-a+1)\mu_i + (n-b+1)\mu_j \\
& - (n-b+1)\mu_i - (n-a+1)\mu_j \\
= \; & (n-a+1)(\mu_i - \mu_j) + (n-b+1)(\mu_j - \mu_i) \\
= \; & (b-a)(\mu_i - \mu_j)
\end{aligned}$$

where $meanFTContrib_{rest}$ is the expected flowtime contributions from the jobs other than i and j, which is the same in both X and X'.
Because in assumption $\mu_i > \mu_j$ and $b > a$, $(b-a)(\mu_i - \mu_j) > 0$, that is:

$$mean(flowtime) > mean(flowtime)'.$$

Similarly, in schedule X, the variance of flowtime contribution from i, $varFTContrib_i$ is $(n - a + 1)^2\sigma_i^2$; the variance of flowtime contribution from j, $varFTContrib_j$, is $(n - b + 1)^2\sigma_j^2$.
For schedule X', $varFTContrib'_i = (n - b + 1)^2\sigma_i^2$ and $varFTContrib'_j = (n - a + 1)^2\sigma_j^2$. Then,

$$\begin{aligned}
& var(flowtime) - var(flowtime)' \\
= \; & varFTContrib_i + varFTContrib_j \\
& - varFTContrib'_i - varFTContrib'_j \\
= \; & (n-a+1)^2\sigma_i^2 + (n-b+1)^2\sigma_j^2 \\
& - (n-b+1)^2\sigma_i^2 - (n-a+1)^2\sigma_j^2 \\
= \; & (2n+2-a-b)(b-a)(\sigma_i^2 - \sigma_j^2)
\end{aligned}$$

Since $\sigma_i^2 > \sigma_j^2$, $b > a$ and $2 \leq a + b \leq 2n$, we get:

$$var(flowtime) > var(flowtime)'.$$

Hence the z value of X is

$$z = \frac{S - mean(flowtime)}{\sqrt{var(flowtime)}}$$

and for X' that is

$$z' = \frac{S - mean(flowtime)'}{\sqrt{var(flowtime)'}}.$$

Clearly, $z < z'$, which means schedule X' has a higher probability to achieve the fixed system performance S than schedule X. That contradicts to X is a β-robust schedule for the n jobs.
Therefore the statement has been proved. □

References

Beck, J. C., and Wilson, N. 2004. Job shop scheduling with probabilistic durations. *Proceedings of the Sixteenth European Conference on Artificial Intelligence* 652–656.

Beck, J. C., and Wilson, N. 2005. Proactive algorithms for scheduling with probabilistic durations. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence* 1201–1206.

Bent, R., and Hentenryck, P. V. 2004. Online stochastic and robust optimization. *Ninth Asian Computing Science Conference* 286–300.

Daniels, R. L., and Carrillo, J. E. 1997. Beta-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions* 29:977–985.

Davenport, A. J.; Gefflot, C.; and Beck, J. C. 2001. Slack-based techniques for robust schedules. *Proceedings of the Sixth European Conference on Planning* 7–18.

Drummond, M.; Bresina, J. L.; and Swanson, K. 1994. Just-in-case scheduling. *Proceedings of the 12thNational Conference on Artificial Intelligence (AAAI)* 1098–1104.

Fowler, D. W., and Brown, K. N. 2003. Branching constraint satisfaction problems and markov decision problems compared. *Annals of Operations Research* 118:85–100.

Gao, H. 1995. Building robust schedules using temporal protectionan empirical study of constraint-based scheduling under machine failure uncertainty. *Masters thesis, Department of Industrial Engineering, University of Toronto, Toronto, Canada*.

Hnich, B.; Smith, B.; and Walsh, T. 2004. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research* 21:357–391.

Walsh, T. 2002. Stochastic constraint programming. *Proceedings of 15th European Conference on Artificial Intelligence* 111–115.

Weisstein, E. W. 2006. Normal distribution function. *Mathworld, Wolfram Research, Inc.* http:// mathworld. wolfram. com/ Normal Distribution Function. html.

Fix the Schedule or Solve Again? Comparing Constraint-Based Approaches to Schedule Execution

Riccardo Rasconi*, Nicola Policella and Amedeo Cesta
Planning & Scheduling Team
ISTC-CNR
Rome, Italy
name.surname@istc.cnr.it

Abstract

This paper discusses alternative approaches to the execution, monitoring and repairing of pre-defined schedules enforced in a constraint-based framework. After presenting a platform able to perform reproducible experiments, we try to shed light on a set of basic trade-offs which focus on two aspects: (a) the type of initial solution generated from an off-line problem solving phase, which represents a baseline for execution. In the attempt to increase proactive robustness, we analyze two alternatives to the classical fixed time schedule – flexible schedules containing a single point solution and partial order schedules. (b) The re-scheduling policy to react to unexpected events: we distinguish between the incremental modification of the initial schedule vs. the retraction of previously made decisions, followed by a new resolution. An interesting set of results is presented then analyzed. Part of the outcome represents a direct confirmation of theoretical expectations from previous analysis; yet, the presence of counterintuitive insights in the results opens the way for further investigation and new perspectives.

Introduction

Despite its relatively recent development, research on schedule execution management is receiving increasing attention. In fact the dynamism and unpredictability which inherently permeate real-world application domains, make the ability to cope with unexpected events during the schedule execution phase an absolutely primary concern.

The growing attention dedicated to this specific issue in research areas such as OR and AI is proved by the increasing number of single results and surveys (Davenport & Beck 2000; Aytug *et al.* 2005; Herroelen & Leus 2004; Verfaillie & Jussien 2005). Notwithstanding the relatively recent developments, the whole topic still offers much room for investigation.

Our aim is to compare different approaches to schedule execution in a fair and controlled way, specifically focusing on project scheduling problems (Brucker *et al.* 1999). These problems are characterized by a rich initial structure: they are based on a network of activities, among which it is possible to identify precedence relations as well as complex temporal relations. As a further source of complexity, several heterogeneous resources with different capacities serve the activities according to complex modalities.

Having identified this problem, in a recent work we have discussed an approach for the generation of unexpected events to develop reusable benchmarks which could be used to assess the efficacy of re-scheduling policies through reproducible experiments (Policella & Rasconi 2005). In this paper we present a set of experiments carried out to compare different approaches to the problem of project scheduling with uncertainty.

Figure 1 shows the complete platform that we have set up in order to produce the experiments. It is composed of three blocks: the *solver* and the *generator* work off-line and have the job of, respectively, computing the initial solution and generating the exogenous events, intended to disturb the schedule execution; the third block, called *SEaM*, *Schedule Execution and Monitoring* (Rasconi, Policella, & Cesta 2006), works on-line, and is responsible of performing a complete simulation of the execution of the initial solution (the baseline schedule). The disturbing events synthesized by the *generator* are injected during the simulated execution at specified times, and their effects are counteracted by the *SEaM* module, which is endowed with a portfolio of *rescheduling* algorithms to the aim of restoring schedule consistency whenever necessary.

The platform is devised to be modular and reusable. By plugging in different versions of either off-line solvers and/or on-line reschedulers (see the dotted boxes in the figure) it is possible to explore different aspects of the execution problem. This paper provides a first analysis on the experiments produced within this schema. In particular: (a) the exogenous events produced by the *generator* are exclusively of temporal nature (e.g., delays in the activities start times, lengthenings in the activities durations, etc.); (b) the solvers are chosen so as to produce initial solutions characterized by different levels of temporal flexibility; (c) the designated set of rescheduling algorithms follow a constraint-based approach.

The analysis of the experimental reports performed in this paper aims to broadening our understanding about the relation between the structural properties of the baseline schedule and the policies chosen for rescheduling, in terms of

*PhD student at the Dipartimento di Informatica, Sistemistica e Telematica, University of Genova.

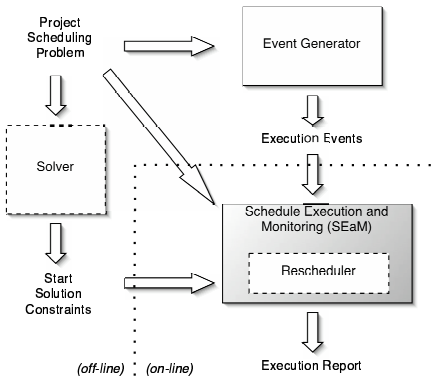


Figure 1: The platform for comparing execution strategies

overall executional behavior. In particular, we want to assess how the various combinations of the two aspects affects execution performance by measuring the number of successfully completed executions, the average worsening of the makespan, etc.

Project Scheduling Problems

The scheduling problem is primarily concerned with finding a suitable temporal allocation for all the tasks that have to be executed so that the final solution guarantees "good" performance relatively to the optimization of an objective function. As already stated, we focus here on a particular family of problems known as *Project Scheduling* (Brucker *et al.* 1999), composed of the following elements:

- **Activities**. $A = \{a_1, \ldots, a_n\}$ represents the set of activities or tasks. Every activity is characterized by a processing time p_i;
- **Resources**. $R = \{r_1, \ldots, r_m\}$ represents the set of the resources necessary for the execution of the activities. Execution of each activity a_i can require an amount req_{ik} of resource r_k to be processed. Different kinds of resources can be taken into account: disjunctive or cumulative, renewable or consumable, among others;
- **Constraints**. The constraints are rules that limit the possible allocations of the activities. They can be divided into two types: (1) the *temporal constraints* impose limitations on the times in which activities can be scheduled. For instance, a binary constraint can be imposed between two activities, in order to mutually bind the instant of occurrence of their start times. Such constraints are usually formulated as bound on differences (Dechter, Meiri, & Pearl 1991). (2) the *resource constraints* limit the maximum capacity of each resource. At no time, the total demand level of any resource being assigned to one or more activities can exceed its maximum capacity.

A classical solution to a project scheduling problem is a *fixed-time schedule* which consists in an assignment of start times to activities $a_1, a_2, \ldots a_n$, i.e. a vector $S = (s_1, s_2, \ldots, s_n)$ where s_i denotes the start time of activity a_i. The time at which activity a_i has been completely processed is called its *completion time* and is denoted by e_i.

This class of problems is interesting because it allows to model a quite broad range of real domains, that is, all the domains that require the modeling of causal relations between activities, coordination between multiple steps, and are characterized by a rich variety of time and resource constraints. In the following paragraph we describe the specific aspects of the reference scheduling problem.

RCPSP/max. The problem we focus upon is the Resource-Constrained Project Scheduling Problem with minimum and maximum time lags, or RCPSP/max (Bartusch, Mohring, & Radermacher 1988).

What distinguishes RCPSP/max instances from other scheduling problems is the presence of temporal constraints which designate either minimum and maximum time lags between the start times of any two activities,

$$l_{ij}^{min} \leq s_j - s_i \leq l_{ij}^{max} \quad (1)$$

where l_{ij}^{min} and l_{ij}^{max} are the minimum and maximum time lag of activity a_j relative to a_i. Moreover, since the processing times are deterministic and preemption is not permitted, completion times are determined by:

$$e_i = s_i + p_i \quad (2)$$

A schedule $S = (s_1, s_2, \ldots, s_n)$ is *time feasible*, if all inequalities given by the activity precedences/time lags (1) and durations (2) hold for start times s_i.

In the case of RCPSP/max we have reusable resources, i.e. they are released when no longer required by an activity and are therefore available for use by another activity. Each activity a_i requires of the use of req_{ik} units of the resource r_k during its processing time p_i. Each resource r_k has a limited capacity of c_k units. A schedule is *resource feasible* if at each time t the demand for each resource $r_k \in R$ does not exceed its capacity c_k, i.e.

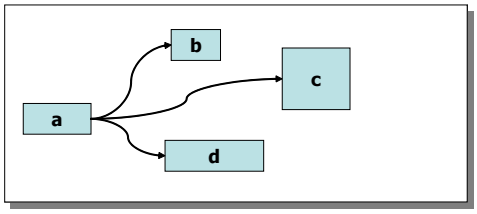
$$\sum_{s_i \leq t < e_i} req_{ik} \leq c_k. \quad (3)$$

A schedule S is said to be *feasible* if it is both time and resource feasible.

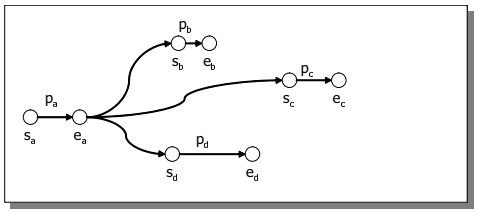
We conclude remarking that RCPSP/max is recognized as a quite complex problem; in fact, even the feasibility version of the problem is NP-hard. The reason for the NP-hardness lies in the presence of maximum time-lags, which inevitably imply the satisfaction of deadline constraints.

Flexible schedules vs. $\mathcal{POS}$s

In this paper we follow a particular research trend with respect to solving scheduling problems, that is based on the general concept of "temporal flexibility". This approach, introduced in (Cheng & Smith 1994) for problems with binary resources and then extended to more general problems in an amount of later works, is based on the fact that the relevant events on a scheduling problem can be represented in a temporal CSP, usually called Temporal Constraint Network



(a) Scheduling problem



(b) Temporal constraint network

Figure 2: Example of a scheduling problem and of its TCN representation.

(TCN) (Dechter, Meiri, & Pearl 1991). The variables in a temporal CSP represent the time points, which can be constrained one another by binding the distance between any two variables. Every activity is associated with two time points (the start and the end times), and by constraining the time points it is possible to define simple precedence relations among activities, task durations, and general separation constraints.

An example is given by Fig. 2 where it is shown a scheduling problem with a resource of capacity 2 and four activities {a,b,c,d}, which require respectively {1,1,2,1} resource units during their execution (note the double height of activity c, representing the higher resource demand). A network of activities is used to describe precedence constraints between pair of activities $\{a \prec b, a \prec c, a \prec d\}$. Moreover we might have a resource conflict among the activities {b,c,d}. Figure 2(b) presents the TCN representation of the scheduling problem. Each activity is represented by two time points the start and the end points. Beyond the precedence constraints present in Fig. 2(a), Fig. 2(b) shows further constraints between pairs of start and end times which model the activity durations.

The search schema used in this approach focus on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pair of activities until all conflicts in the current problem representation are removed. This approach is usually referred to as the Precedence Constraint Posting, PCP (Cheng & Smith 1994), because it revolves around imposing precedence constraints to solve the resource conflicts, rather than fixing rigid values to the start times.

In (Cesta, Oddi, & Smith 1998) it is shown that though the previous schedule representation inherently provides a certain level of flexibility at execution time, it guarantees a solution only if lower bounds of the feasibility intervals are chosen for the time points, as described in the following definition:

Definition 1 (Flexible schedule) *A flexible schedule for a problem $\mathcal{P}$ is a network of activities, (readily interpretable as a temporal graph), such that a feasible solution for the problem is obtained by allocating each activity at the temporal lower bound allowed by the network.*

In order to overcome the limitation imposed by the flexible schedule, i.e. having only one consistent solution, a generalization of the TCN produced by a PCP phase is proposed in works such as (Cesta, Oddi, & Smith 1998; Policella *et al.* 2004), in which methods for defining a set of both time and resource feasible solutions are presented. This new representation is called *Partial Order Schedule*:

Definition 2 (Partial Order Schedule) *A Partial Order Schedule $\mathcal{POS}$ for a problem $\mathcal{P}$ is an network of activities, such that any possible temporal solution is also a resource-consistent assignment.*

A $\mathcal{POS}$ is a special case of a flexible solution. With respect to the solution synthesized by the PCP phase as shown in the cited works, a $\mathcal{POS}$ can be obtained replacing the *solution constraints* with a new set of constraints that impose a stronger condition on the TCN. For reasons that will be explained later, we call this new set of constraints *chaining constraints*. By adding the *chaining constraints* to the problem constraints, we have a particular TCN that identifies not only a set of temporal solution but also a set of resource feasible solutions.

It is worth underlining that in existing literature the "goodness" of different flexible solutions has been measured *a priori* with structural measures, called flexibility and fluidity in (Policella *et al.* 2004), on the resulting TCN; the present work offers a first experimental comparison of fixed time solutions, flexible schedules and $\mathcal{POS}$s, with respect to a homogeneous set of benchmarks. This analysis offers the possibility to confirm (or deny!) theoretical conjectures from a truly empirical standpoint.

Fixing the Schedule or Solving Again

Execution uncertainty in scheduling can be dealt with either *proactively*, or *reactively*. In the proactive approach, the solving process is performed taking into account optimization functions that consider execution robustness as a metric to evaluate the produced solutions; in the reactive approach, uncertainty is not taken into account during the construction of the initial solution: all efforts aimed at counteracting the effects of unforeseen events are postponed to the on-line phase, where the schedule is adjusted every time a change compromises proper execution. This approach is obviously based on the implicit assumption that generally it is not possible to bound the scope of a change required to the current schedule in advance.

The key point in the proactive approach is to synthesize solutions that are able to absorb the effects of unexpected events without rescheduling, while the reactive approach requires to exploit local or global adjustments to the schedule, in order to re-gain consistency. As it will be shown, the boundary between these two strategies is often very thin; the objective of this paper is to investigate the possibilities offered by trading-off among simple adjustments, fixes to the

baseline schedule, and new calls to problem solvers. To this aim, we have built an experimental environment targeted at studying how increasingly robust baselines can influence the reactive phase.

Local vs. Global Reactivity. In the current research scenario it is possible to distinguish two different reactive scheduling approaches: local and global approaches. An example of local approach is represented by OPIS (Smith 1994), a scheduling system designed to incrementally revise schedules in response to changes in the solution constraints. This approach is based on an analysis of the schedule flaws which may be caused by the occurrence of unpredicted events; at each perturbation, the system initiates a cycle in which the outstanding conflicts are iteratively repaired by triggering local modifications, until consistency is restored. On the other hand, the approach based on the resolution of the minimal perturbation problem (El Sakkout & Wallace 2000), pursues a global approach to obtain the same goal. This in general guarantees to find a higher quality solution with respect to a local method. Even though solution quality is an extremely relevant issue, the global approach exhibits a lack in reactivity with respect to the local approach, as it requires more time for system reconfiguration.

In our schema, we model the local and global approach to rescheduling by enabling different constraint removal strategies on the current solution[1]. More specifically:

– *No-Retraction strategy*: in a local perspective, before each revision, none of the constraints imposed is removed, and the subsequent solution is computed by adding further solution constraints.

– *Retraction strategy*: in a global perspective, before each revision, all the previously imposed solution constraints are removed; the subsequent solution is computed by adding new constraints to this "clean" representation.

We notice that, because of the min/max time lags (1), either the strategies do not always solve all the events that may occur – in fact max time lags imply deadline constraints on the activities: for this reason the No-retraction strategy can produce solutions more constrained (hence more fragile) than the ones produced by the Retraction strategy. On the other hand the Retraction strategy will require more CPU time than the No-retraction alternative.

Uncertainty in Schedule Execution. In the present study, we focus our attention on the temporal changes which normally characterize the physical environments. In particular, we produce a set of exogenous events for every simulated schedule execution. For obvious reasons, each set is computed on the basis of the baseline schedule's initial characteristics. For the present analysis, we limit ourselves to the generation of temporal events, such as delays of the activities start times and/or modifications of activity processing times:

– *delay of the activity start time*: activity a_i undergoes a delay of Δ_{st} time units, at $t = t_{aware}$ ($e_{delay} = \langle a_i, \Delta_{st}, t_{aware} \rangle$);

– *change of activity processing time*: activity a_i's processing time p_i is extended by Δ_p time units, at t_{aware} ($e_p = \langle a_i, \Delta_p, t_{aware} \rangle$).

Both the events are characterized by the temporal value t_{aware}. This specifies the temporal instant in which, during the execution, we are aware of the presence of a given event. During the event generation phase this value is computed in order to be always smaller than the start time of the influenced activity in the case of delay events and smaller of the end time in the case of changes of processing time.

For reasons of space we cannot give a complete account on the event generation here. The reader should refer to (Policella & Rasconi 2005) for further details.

Algorithm 1: Solve a scheduling problem P and Execute one of its solution S

```
Input: problem P, policies parameter retract and pos
Output: Execution report
// off-line phase
S ← offlineScheduler(P)
if S does not exist then
    STOP (SOLVER FAILURE)
if pos then
    S ← createPOS(S)
// on-line phase
while a disturb E exists do
    if retract then
        if propagation(E, S) fails ∨ S is not resource consistent then
            S ← removeChoice (S)
            if propagation(E,S) fails then
                STOP (EXECUTION FAILURE)
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
            if pos then
                S ← createPOS*(S)
    else
        if propagation(E,S) fails then
            STOP (EXECUTION FAILURE)
        if S is not resource consistent then
            S ← onlineScheduler(S)
            if S does not exist then
                STOP (EXECUTION FAILURE)
```

Integration of Proactive and Reactive Phases

The core of our work, see Fig. 1, is based on the execution module we call *SEaM* (for Scheduling Execution and Monitoring). This component integrates the management of the

[1]Exception is made for the constraints which model the dynamic aspects of the progressing execution. These constraints are, of course, always preserved.

following objects: (1) the scheduling problem, (2) the initial solution, (3) a world simulation in terms of temporally spaced exogenous events, and (4) the repair strategies.

Algorithm 1 puts together off-line and on-line work. The differentiation between the off-line (proactive) and on-line (reactive) phases is immediately visible. In order to produce flexible schedules (`offlineScheduler()` in the algorithm), we have implemented the ISES procedure described in (Cesta, Oddi, & Smith 1999), because this algorithm has proved to be effective on RCPSP/max problems. It relies, at its core, on a PCP search procedure, which generates flexible schedules by incrementally removing resource conflicts from a temporally feasible solution. This search is then embedded within a larger iterative-sampling framework to broaden search space coverage and promote solution optimization.

$\mathcal{POS}$s generation (`createPOS()`), is performed by applying a procedure introduced in (Policella *et al.* 2004) called CHAINING, to a flexible schedule previously obtained with ISES. In particular, in the off-line phase we use the iterative-sampling version of this procedure, so as to enhance the robustness and stability properties of the produced solution.

At execution time, we privilege algorithms that are not too demanding in term of CPU time. For this reason we slightly modified the previous ISES and CHAINING procedures to obtain computationally lighter versions denoted, respectively, with `onlineScheduler()` and `createPOS*()`. In fact the original procedures, being based on an iterative schema, are CPU expensive: the modified versions are therefore implemented so as to stop the computation as soon as a first viable solution is found, in the aim of reducing the computational effort.

Moreover, it is worth noting that the `createPOS*()` procedure is called into play during execution only if we are enacting the Retraction strategy. In fact, in the No-Retraction case, the initial $\mathcal{POS}$ preserves all its characteristics even if new constraints are posted in the problem, and there is no need to launch the chaining procedure again.

In order to distinguish among all the different execution combinations, we introduce two flags in the algorithm:

– the flag `pos` allows to distinguish the case in which a $\mathcal{POS}$ is created (case POS), from the case in which a flexible schedule is used (case FS);

– the flag `retract` allows to distinguish between the Retraction strategy (case R from "Retract") and the No-Retraction strategy case (NR).

From this distinction we globally recognize the four strategies that will be compared in the experimental analysis, and denoted as: FS-NR, FS-R, POS-NR and POS-R.

In order to further clarify the algorithm, the following points should be remarked: (a) regardless of how the initial solution is computed (flexible schedule or $\mathcal{POS}$), the on-line phase of the algorithm is always initiated with the earliest start time solution; (b) the `propagation()` function represents a call to the temporal propagation on the TCN; (c) as the execution proceeds, further constraints are posted in the problem to model the actual start of an activity, i.e., the constraints which fix the start time of an activity to a precise time instant.

Example. Figure 3 shows the two phases, off-line and on-line, of the scheduling problem described in Fig. 2. In the off-line phase two solutions are produced: a partial order schedule (left hand side) and a flexible solution (right hand side).

In the on-line phase we simulate the lengthening of the activity b duration. This change is represented by changing the constraint that models the duration of activity b: from p_i to $new\ p_i$. In the left hand side is shown how the $\mathcal{POS}$ is able to react to the disturb without any repair. In fact, due to its inherently constrained structure, the chances to respond to the exogenous event without the need to reschedule are maximized.

On the right hand instead we have the execution of the flexible schedule. As said before, a problem change can be faced by using two alternative strategies. Using the No-retraction strategy we obtain a new solution by posting a new precedence constraint between b and c. In the case of the Retraction strategy we see that as soon as the new event is acknowledged, the precedence between d and c is removed (the previously imposed solving constraint), and a new solution is found by imposing a precedence between b and c.

Experimental Analysis

The comparison presented in this section is based on a scheduling problem benchmark taken from OR community, $j100$ (Kolisch, Schwindt, & Sprecher 1998). This consists of 540 scheduling problem instances each of 100 activities and 5 resources. Each problem belonging to the $j100$ scheduling benchmark is executed with four instances of world simulations of different size (1, 2, 3, and 5 events each). Each instance of the reactive scheduling benchmark is composed of a set of properly modeled disturbing events. As stated above, each event represents either a delay on the start time, or a delay on the end time of the activities. These two event types are produced with the same probability.

It is worth remarking at this point that not all the problems belonging to the scheduling benchmark admit a solution; moreover, the off-line scheduler, ISES, does not perform a systematic search. For these reasons, our experimental analysis is based on a subset of 480 predictively solved problems. Moreover, the execution of a project scheduling problem can not be considered completed until all the activities are successfully processed. This represents a major difference with respect to other dynamic problems where the execution of certain activities can be overruled.

Results. Table 1 shows the results of our investigation. As explained earlier, we evaluate the different combinations of off-line/on-line policies — POS-R, POS-NR, FS-R, and FS-NR. To make the comparison more complete, we add a further execution mode based on the use of fixed time solution where each activity is assigned a single start time instead of a set of alternatives. For each entry in the tables, we take into account the following aspects:

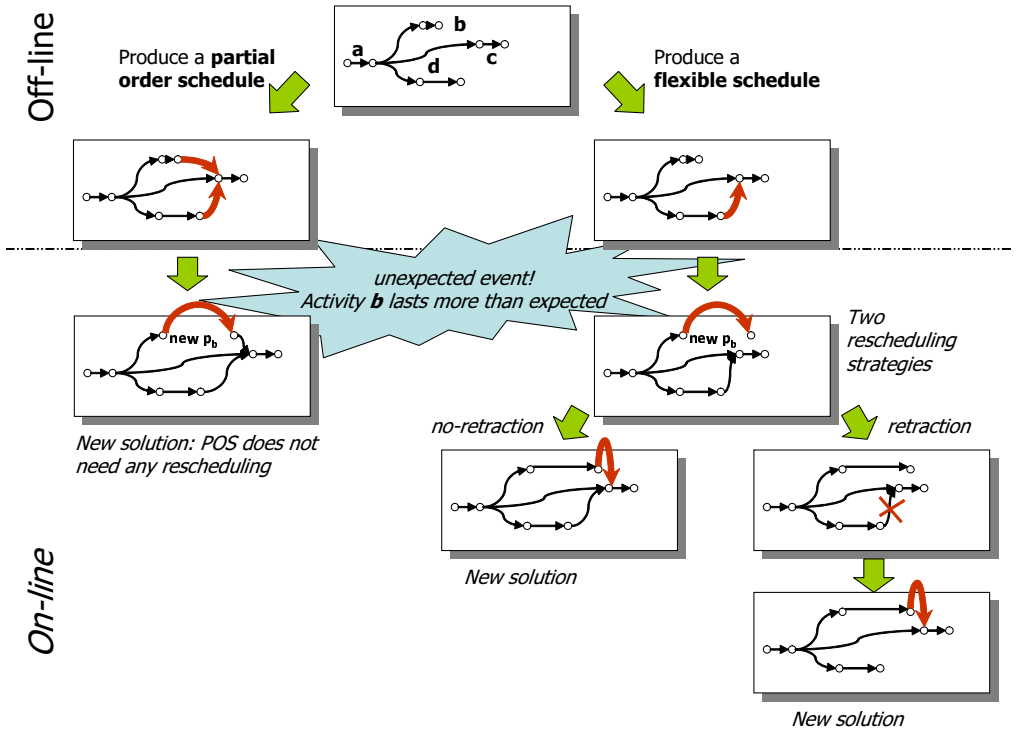


Figure 3: Comparison of different schedule shapes execution

– the number of disturbs (unexpected events) injected during each single execution;

– percentage (with respect to the number of initially solved problems) of the schedules which successfully completed the execution (*% executed*);

– execution failure percentage due to the inability to find an alternative solution (*% failed resch.*);

– execution failure percentage due to the impossibility to accept the exogenous event on behalf of the TCN (*% refused events*);

– the average makespan of the solutions at the end of the execution (*mk*);

– the average difference between the initial and the final makespan (Δ *mk*);

– the percentage of the performed rescheduling actions with respect to the number of the injected disturbs (*% rescheduling*). We recall that we have a rescheduling action each time the on-line solver is invoked;

– the average CPU time, in msecs, to compute the initial solution (*CPU Off-line*);

– the average CPU time spent to perform all reschedulings during the execution (*CPU On-line*);

– the sensitivity of activity start time w.r.t. the execution process (ψ):

$$\psi = \sum_{i=1}^{N} \frac{|s_i^f - s_i^0|}{N}$$

where s_i^0 and s_i^f are respectively the initial start time of a_i and the start time in which the activity is really started. This measure gives an assessment of how much the initial solution has been affected by the occurrence of the exogenous events during the execution. The lower the value, the more the solution proved to be stable.

It is worth remarking that the execution can terminate with failure for two alternative reasons: (1) the exogenous event produces a temporal inconsistent situation and (2) the event is consistent with the current situation but the on-line solver is not able to find an alternative solution (also in the execution phase we do not use a systematic search). The two situations are reported respectively in column *% refused events* and column *% failed resch.* Notice that during the execution, where further constraints are added to simulate the begin of an activity, it is possible to have a refused event also in the case the Retraction strategy is used. Finally we specify that, for a fair comparison of the different policies, the data presented in the rightmost part of the table are computed on the basis of the problem instances commonly executed with all the execution strategies.

Comments. One of the most striking results that we observe regards the different abilities in preserving the executability of a solution. The outcome shows that the use of partial order schedules tends to lower the success rate in terms of completed executions (*% executed* column). As the table presents, this is mainly due to the dramatic increase in the number of rejected disturbs (*refused events* column). This apparent anomaly can be explained as follows: the cre-

	number of disturbs	% executed	failed resch.	refused events	mk	Δmk	% resched.	CPU off-line	CPU on-line	ψ
FS-R		91,04%	2,08%	6,88%	424,60	9,02	24,38%	36242,48	766,15	5,44
POS-R	**1**	87,29%	2,08%	10,63%	419,88	5,07	11,58%	36287,86	303,97	3,00
FS-NR		91,87%	1,25%	6,87%	419,06	3,48	24,14%	36242,48	130,74	1,48
POS-NR		86,87%	2,50%	10,62%	417,11	2,31	11,58%	36287,86	54,59	1,05
fixed time		89,79%	3,75%	6,45%	437,36	21,78	99,75%	36242,48	3035,68	15,16
FS-R		85,21%	3,13%	10,66%	435,54	13,95	23,04%	30259,15	874,70	9,48
POS-R	**2**	76,46%	2,29%	21,25%	429,22	8,41	10,03%	32371,82	674,86	5,17
FS-NR		85,62%	2,71%	11,66%	427,90	6,30	22,88%	30259,15	258,50	3,07
POS-NR		73,95%	5,00%	21,04%	424,74	3,93	9,56%	32371,82	97,46	2,02
fixed time		81,25%	8,54%	10,20%	446,62	25,02	99,53%	30259,15	2768,37	17,73
FS-R		79,17%	3,96%	16,87%	449,84	19,40	20,27%	26318,37	963,16	11,83
POS-R	**3**	69,58%	2,92%	27,50%	441,49	11,93	9,41%	28406,98	675,18	6,96
FS-NR		80,00%	2,50%	17,50%	439,66	9,23	22,37%	26318,37	371,03	4,08
POS-NR		67,71%	5,41%	26,87%	436,24	6,68	9,63%	28406,98	151,26	3,16
fixed time		77,50%	6,87%	15,62%	458,32	27,89	99,22%	26318,37	3716,15	19,77
FS-R		70,21%	4,17%	25,63%	464,12	28,85	22,45%	25682,40	2391,92	17,36
POS-R	**5**	60,42%	3,13%	36,46%	455,56	21,07	10,57%	27544,98	1748,56	13,12
FS-NR		70,41%	3,33%	26,25%	447,42	12,15	21,66%	25682,40	646,90	5,80
POS-NR		56,66%	7,08%	36,25%	444,68	10,18	10,48%	27544,98	289,17	4,81
fixed time		67,08%	8,33%	24,58%	465,56	30,29	98,43%	25682,40	6721,48	19,74

Table 1: Summarizing data for each execution strategy (the values in the last six columns are computed on the intersection set of all successfully executed j100 problems)

ation of a $\mathcal{POS}$ inherently involves a higher level of "constrainedness" in the TCN, in order to guarantee a resource conflict-free solution. This circumstance inevitably makes the TCN more reluctant in accepting new contraints, in the specific case, the constraints which model the exogenous events. Also, note how this effect gets worse as the number of the exogenous events increases (86,87% in the POS-NR case with 1 event, against 56,66% with 5 events).

It should be noted how the highest rate of failed reschedulings (*failed resch.* column) is reached in the execution of *fixed time* solutions. This result is not surprising, as fixed time solutions require a rescheduling every time an exogenous event occurs; since the rescheduler does not perform a complete search, the chance of failure gets sensibly higher. It can again be noticed how, on average, this effect grows with the number of injected events.

The rightmost part of Table 1 offers different yet interesting results. One of the most important characteristic to be observed is the extremely low rate of necessary reschedulings exhibited by the POS-R/POS-NR policies (*% resched.* column): this result is all but surprising and confirms the theoretical expectations which motivated the study on the $\mathcal{POS}$. As shown, the need for schedule revision in case of $\mathcal{POS}$ utilization roughly decreases by more than 50% in case of 5 disturbs. Note also the $\approx 100\%$ *reschedulings* figure relative to the case of *fixed time* schedules: in this case, a schedule revision is practically always needed: this is confirmed by the extremely high *CPU on-line* values.

A maybe misleading results is given by the comparison of the final makespan (mk) obtained respectively by using the Retraction and the No-Retraction strategies. In fact, one would expect the R strategies (which allow a greater re-shuffling) to return better makespan values with respect to NR strategies. This is not our case because, as described before, the rescheduling actions are performed by using a less specialized makespan-optimizing procedure (`onlineScheduler` in Algorithm 1) which tends to spoil the makespan quality. On the other hand, the NR strategy that tries to maintain the schedule continuity is also able to obtain a preservation of makespan values.

Another interesting aspect can be observed by comparing the *CPU on-line* values between the *Retraction* and *No Retraction* strategies. In general, the Retraction methods require a higher CPU on-line load because the removal of the solution constraints inevitably re-introduces some resource conflicts that must be solved by rescheduling. But the intriguing result lies in the fact that this difference in the CPU on-line rates stands *despite the comparable amount of performed reschedulings*. Let us look at the difference between the FS-R and FS-NR rates: it can be seen that, in the 5 events case, we have 2392 ms. (FS-R) against 647 ms. (FS-NR), although the number of performed reschedulings is practically the same ($\approx 21\%$)! The same effect can be observed between the POS-R and POS-NR cases: 1748 ms. against 289 ms, notwithstanding the same ($\approx 10.5\%$) number of reschedulings. This circumstance can be explained as follows: NR execution modes retain all the temporal constraints of the previous solution: hence, the rescheduler is bound to work on a smaller search space, finding the next solution almost immediately.

One last word on the ψ column: these values can in fact be taken as a measure of the solution stability. As it can be

seen, the highest stability during execution is obtained with the NR methodologies, as they maintain the structure of the initial solution for the whole execution. The most unstable solutions are produced in the fixed time case, as a direct consequence of the complete lack of temporal flexibility.

Conclusions and Future Work

This paper discusses alternative approaches to the execution, monitoring and repairing of pre-defined schedules enforced in a constraint-based framework – *SEaM*. This framework allows us to obtain different insights on the combination of two aspects: (a) the use of temporal flexible solutions to increase proactive robustness and (b) the use of complementary re-scheduling policies to react to unexpected events. In particular we analyze two alternatives to the classical fixed time schedule – flexible schedules, containing a single point solution, and partial order schedules or $\mathcal{POS}$s. Moreover, we distinguish between the incremental modification of the initial schedule vs. the retraction of previously made decisions followed by a new resolution.

A set of results support the usefulness of this analysis. Part of the outcome represents a direct confirmation of theoretical expectations from previous analysis; yet, the presence of counterintuitive insights in the results opens the way for further investigation and new perspectives.

For future work we have several alternatives to pursue. First we want to extend this analysis to different benchmarks both in terms of modeled disturbs and reference scheduling problems (either problems of larger size or different families of problems).

A different point consists, instead, of enriching the *SEaM* framework with further repair strategies: for instance we want to introduce complete search methods (i.e., branch-and-bound techniques) to evaluate the trade-off between the efficiency and computational complexity with respect the heuristic method used in this paper. Another opportunity could be to enrich the chaining procedure with de-constraining methods like the one introduced in (Muscettola, Morris, & Tsamardinos 1998). In fact, being a $\mathcal{POS}$ a temporal network, it is possible to apply these reformulation methods to reduce the constrainedness of the produced $\mathcal{POS}$s and further speed up the propagation phase.

Additionally, we also aim at modifying the solutions considered for the execution. As remarked above, for both flexible and partial order schedules we consider the earliest start time solution. In fact, a different solutions could provide a certain degree of temporal and/or resource redundancy that, if intelligently distributed, might reveal useful to better tolerate unforeseen events.

Acknowledgments. The authors' work is partially supported by MIUR (Italian Ministry for Education, University and Research) under projects ROBOCARE (L. 449/97) and "Vincoli e Preferenze" (PRIN).

References

Aytug, H.; Lawley, M. A.; McKay, K. N.; Mohan, S.; and Uzsoy, R. M. 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research* 165(1):86–110.

Bartusch, M.; Mohring, R. H.; and Radermacher, F. J. 1988. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research* 16:201–240.

Brucker, P.; Drexl, A.; Mohring, R.; Neumann, K.; and Pesch, E. 1999. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research* 112:3–41.

Cesta, A.; Oddi, A.; and Smith, S. F. 1998. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the* 4^{th} *International Conference on Artificial Intelligence Planning Systems, AIPS-98*, 214–223. AAAI Press.

Cesta, A.; Oddi, A.; and Smith, S. F. 1999. An Iterative Sampling Procedure for Resource Constrained Project Scheduling with Time Windows. In *Proceedings of the* 16^{th} *International Joint Conference on Artificial Intelligence*, 1022–1029. Morgan Kaufmann.

Cheng, C., and Smith, S. F. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings of the* 12^{th} *National Conference on Artificial Intelligence, AAAI-94*, 1086–1091. AAAI Press.

Davenport, A. J., and Beck, J. C. 2000. A Survey of Techniques for Scheduling with Uncertainty. accessible on-line at http://tidel.mie.utoronto.ca/publications.php on Feb, 2006.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

El Sakkout, H. H., and Wallace, M. G. 2000. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints* 5(4):359–388.

Herroelen, W., and Leus, R. 2004. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research* 42(8):1599–1620.

Kolisch, R.; Schwindt, C.; and Sprecher, A. 1998. Benchmark Instances for Project Scheduling Problems. In Weglarz, J., ed., *Project Scheduling - Recent Models, Algorithms and Applications*. Boston: Kluwer Academic Publishers. 197–212.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the* 6^{th} *International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*.

Policella, N., and Rasconi, R. 2005. Designing a testset generator for reactive scheduling. *Intelligenza Artificiale (Italian Journal of Artificial Intelligence)* 2(3):29–36.

Policella, N.; Oddi, A.; Smith, S. F.; and Cesta, A. 2004. Generating Robust Partial Order Schedules. In *Principles and Practice of Constraint Programming,* 10^{th} *International Conference, CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, 496–511. Springer.

Rasconi, R.; Policella, N.; and Cesta, A. 2006. SEaM: Analyzing Schedule Executability through Simulation. In *The* 19^{th} *International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE'06)*.

Smith, S. F. 1994. OPIS: A Methodology and Architecture for Reactive Scheduling. In Fox, M., and Zweben, M., eds., *Intelligent Scheduling*. Morgan Kaufmann.

Verfaillie, G., and Jussien, N. 2005. Constraint solving in uncertain and dynamic environments – a survey. *Constraints* 10(3):253–281.

Distributed Constraint Satisfaction Problems to Model Railway Scheduling Problems

M. Abril, M. A. Salido, F. Barber, L. Ingolotti, P. Tormos, A. Lova
Dpto. Sistemas Informáticos y Computación
Camino de Vera s/n
46071, Valencia, Spain
{mabril, msalido, fbarber, lingolotti}@dsic.upv.es
{ptormos, allova}@deio.upv.es

Abstract

Railway Scheduling is considered to be a difficult and time-consuming task. This is due to real railway networks can be modelled as Constraint Satisfaction Problems (CSPs), but they require a huge number of variables and constraints. The general CSP is known to be NP-complete; however, distributed models may reduce the exponential complexity by dividing the problem into a set of subproblems. In this work, we present several proposals to distribute the railway scheduling problem into a set of sub-problems as independent as possible. The first technique carries out a partition over the constraint network, meanwhile the second distributes the problem by trains and the third technique divides the problem by means of contiguous stations.

Introduction

Train timetabling is a difficult and time-consuming task, particularly in the case of real networks, where the number of constraints and the complexity of constraints grow drastically. A feasible train timetable should specify the departure and arrival time of each train to each location of its journey, in such a way that the line capacity and other operational constraints are taken into account. Traditionally, train timetables are generated manually by drawing trains on the time-distance graph. The train schedule is generated from a given starting time and is manually adjusted so that all constraints are met. High priority trains are usually placed first followed by lower priority trains. It can take many days to develop train timetables for a line, and the process usually stops once a feasible timetable has been found. The resulting plan of this procedure may be far from optimal.

The literature of the 1960s, 1970s, and 1980s relating to rail optimization was relatively limited. Compared to the airline and bus industries, optimization was generally overlooked in favor of simulation or heuristic-based methods. However, Cordeau et al. (Cordeau, Toth, & Vigo 1998) point out greater competition, privatization, deregulation, and increasing computer speed as reasons for the more prevalent use of optimization techniques in the railway industry. Our review of the methods and models that have been published indicates that the majority of authors use models that are based on the Periodic Event Scheduling Problem (PESP) introduced by Serafini and Ukovich (Serafini & Ukovich 1989). The PESP considers the problem of scheduling as a set of periodically recurring events under periodic time-window constraints. The model generates disjunctive constraints that may cause the exponential growth of the computational complexity of the problem depending on its size. Schrijver and Steenbeek (Schrijver & Steenbeek 1994) have developed CADANS, a constraint programming- based algorithm to find a feasible timetable for a set of PESP constraints. The scenario considered by this tool is different from the scenario that we used; therefore, the results are not easily comparable. Nachtigall and Voget (Nachtigall & Voget 1997) also use PESP constraints to model the cyclic behavior of timetables and to consider the minimization of passenger waiting times as the objective function. Their solving procedure starts with a solution that is obtained in a way similar to the one that timetable designers in railway companies use. This initial timetable is then improved by using a genetic algorithm. In our problem, the waiting time for connections is not taken into account because we only consider the timetabling optimization for a single railway line. The train scheduling problem can also be modeled as a special case of the job-shop scheduling problem (Silva de Oliveira (Silva de Oliveira 2001), Walker et al. (Walker & Ryan 2005)), where train trips are considered as jobs that are scheduled on tracks that are regarded as resources. The majority of these works consider the scheduling of new trains on an empty network. However, railway companies usually also require the optimization of new trains on a line where many trains are already in circulation (that is, trains that have a fixed timetable). With this main objective, Lova et al. (Lova *et al.* 2006) propose a scheduling method based on reference stations where the priority of trains, in the case of conflict, changes from one iteration to another during the solving process.

Our goal is to model the railway scheduling problem as a Constraint Satisfaction Problems (CSPs) and solve it using constraint programming techniques. However, due to the huge number of variables and constraints that this problem generates, a distributed model is developed to distribute the resultant CSP into a semi-independent subproblems such as the solution can be found efficiently.

The overall goal of a long-term collaboration between our

group at the Polytechnic University of Valencia (UPV) and the National Network of Spanish Railways (RENFE) is to offer assistance to help in the planning of train scheduling, to obtain conclusions about the maximum capacity of the network, to identify bottlenecks, etc.

In parallel computing, many researchers are working on graph partitioning (Schloegel, Karypis, & Kumar 2003), (Karypis & Kumar 1998). The main objective of these techniques is to divide the graph into a set of regions such that each region has roughly the same number of nodes and the sum of all edges connecting different regions is minimized. Fortunately, many heuristics may solve this problem efficiently. For instance, graphs with over 14000 nodes and 410000 edges can be partitioned in under 2 seconds (Karypis & Kumar 1995). Graph partitioning can also be applied to constraint satisfaction problem. Thus, we can use ideas about graph partitioning, when dealing with railway scheduling problem, to distribute the problem into a set of sub-problems.

In this work, we propose several ways to distribute the railway scheduling problem. It is partitioned into a set of subproblems by means of graph partitioning, by means of types of trains and by means of contiguous constraints.

In the following section, we summarize some definitions. In section 3, we study three models to distribute the railway scheduling problem. In section 4, we present the distributed model to be solved by the DCSP. An evaluation among different models is carried out in section 5. Finally we summarizes the conclusions and future work in section 6.

Definitions

This section presents CSPs in a slightly non-standard form, which will be convenient for our purposes, and will unify works from constraint satisfaction communities.

Definition 1: A *CSP* consists of:

- a set of variables $X = \{x_1, x_2, ..., x_n\}$
- each variable $x_i \in X$ has a set D_i of possible values (its domain)
- a finite collection of constraints $C = \{c_1, c_2, ..., c_p\}$ restricting the values that the variables can simultaneously take.

A solution to a CSP is an assignment of values to all the variables so that all constraints are satisfied; a problem is *satisfiable* or *consistent* when it has a solution at least.

State: one possible assignment of all variables.

Partition : A partition of a set C is a set of disjoint subsets of C whose union is C. The subsets are called the blocks of the partition.

A ***running map*** : contains information regarding railway topology (stations, tracks, distances between stations, traffic control features, etc.) and the schedules of the trains that use this topology (arrival and departure times of trains at each station, frequency, stops, crossings, etc,).

Distributed CSP: A distributed CSP (DCSP) is a CSP in which the variables and constraints are distributed among automated agents (Yokoo & Hirayama 2000).

Each agent has some variables and attempts to determine their values. However, there are interagent constraints and the value assignment must satisfy these interagent constraints. In our model, there are k agents $1, 2, ..., k$. Each agent knows a set of constraints and the domains of variables involved in these constraints.

Definition 2: A *block agent* a_j is a virtual entity that essentially has the following properties: autonomy, social ability, reactivity and pro-activity (Wooldridge & Jennings 1995).

Block agents are autonomous agents. They operate their subproblems without the direct intervention of any other agent or human. *Block agents* interact with each other by sending messages to communicate consistent partial states. They perceive their environment and changes in it, such as new partial consistent states, and react, if possible, with more complete consistent partial states.

Definition 3: A *multi-agent system* is a system that contains the following elements:

1. An environment in which the agents live (variables, domains, constraints and consistent partial states).
2. A set of reactive rules, governing the interaction between the agents and their environment (agent exchange rules, communication rules, etc).
3. A set of agents, $A = \{a_1, a_2, ..., a_k\}$.

Constraints in the Railway Scheduling Problem

There are three groups of scheduling rules in our railway scheduling problem: traffic rules, user requirements rules and topological rules. A valid running map must satisfy the above rules. These scheduling rules can be modelled using the following constraints, where variable $TA_{i,k}$ represents that train i arrives at station k and the variable $TD_{i,k}$ means that train i departs from station k:

1. **Traffic rules** guarantee crossing and overtaking operations. The main constraints to take into account are:
   - ***Crossing constraint***: Any two trains going in opposite directions must not simultaneously use the same one-way track.
     $$TA_{i,A} < TD_{j,A} \text{ or } TA_{j,B} < TD_{i,B}$$
     The crossing of two trains can be performed only on two-way tracks and at stations, where one of the two trains has been detoured from the main track (Figure 1).
   - ***Overtaking constraint***: Any two trains (T_i and T_j) going at different speeds in the same direction can only overtake each other at stations.
     $$TD_{i,A} < TD_{j,A} \rightarrow TA_{i,B} < TA_{j,B}$$
     The train being passed is detoured form the main track so that the faster train can pass the slower one (see Figure 1).
   - ***Expedition time constraint***. There exists a given time to put a detoured train back on the main track and exit from a station.

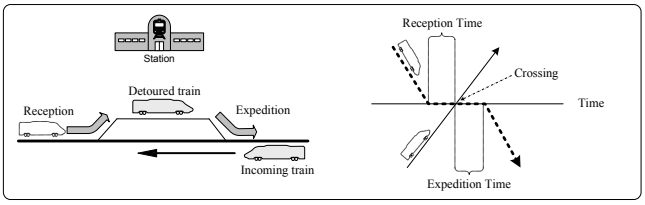


Figure 1: Constraints related to crossing and overtaking in stations

- ***Reception time constraint***. There exists a given time to detour a train from the main track so that crossing or overtaking can be performed.

2. **User Requirements**: The main constraints due to user requirements are:
   - ***Type and Number of trains*** going in each direction to be scheduled.
   - ***Path of trains***: Locations used and ***Stop time*** for commercial purposes in each direction.
   - ***Scheduling frequency***. Train departure must satisfy frequency requirements in both directions. This constraint is very restrictive because, when crossings are performed, trains must wait for a certain time interval at stations. This interval must be propagated to all trains going in the same direction in order to maintain the established scheduling frequency. The user can require a fixed frequency, a frequency within a minimum and maximum interval, or multiple frequencies.
   - ***Departure interval*** for the departure of the first trains going in both the up and down directions.
   - ***Maximum slack***. This is the maximum percentage δ that a train may delay with respect to the minimum journey time.
3. **Railway infrastructure Topology and type of trains** to be scheduled give rise to other constraints to be taken into account. Some of them are:
   - Number of ***tracks in stations*** (to perform technical and/or commercial operations) and the number of tracks between two locations (one-way or two-way). No crossing or overtaking is allowed on a one-way track,
   - ***Time constraints***, between each two contiguous stations,
   - Added ***Station time constraints*** for technical and/or commercial purposes.

The complete set of constraints, including an objective function, transform the CSP into a constraint satisfaction and optimization problem (CSOP), where the main objective function is to minimize the journey time of all trains. Variables are frequencies and arrival and departure times of trains at stations. Constraints are composed by user requirements, traffic rules, and topological constraints.

The complete CSOP is presented in Figure 2. Let's suppose a railway network with r stations, n trains running in the down direction, and m trains running in the up direction. We assume that two connected stations have only one line connecting them. $Time_{i,k-(k+1)}$ is the journey time of train i to travel from station k to $k+1$; $TS_{i,k}$ and $CS_{i,k}$ represent the technical and commercial stop times of train i in station k, respectively; and ET_i and RT_i are the expedition and reception time of train i, respectively.

Integer Variables $\forall i = 1..n \forall k = 1..r$

$TA_{i,k}$ 0..86400;

$TD_{i,k}$ 0..86400;

Objective Function

(1) **Min** $\sum_{i=1}^{i=n}(TA_{i,r} - TD_{i,1}) + \sum_{j=1}^{j=m}(TA_{j,1} - TD_{j,r})$;

Subject to

/Frequency Constraints $\forall i = 1..n \forall k = 1..r$

(2) $TD_{i+1,k} - TD_{i,k} = Frequency$;

/Time Constraints $\forall i = 1..n \forall k = 1..r$

(3.1) $TA_{i,k+1} - TD_{i,k} = Time_{i,k-(k+1)}$;

(3.2) $TA_{i,k} - TD_{i,k+1} = Time_{i,k-(k+1)}$;

/Stations Time Constraints $\forall i = 1..n \forall k = 1..r$

(4) $TD_{i,k} - TA_{i,k} - TS_{i,k} = CS_{i,k}$;

/Constraints to limit journey time $\forall i = 1..n \forall j = 1..m$

(5.1) $TA_{j,r} - TD_{i,1} \leq (1 + \frac{\delta}{100}) * Time_{i,1-r}$;

(5.2) $TA_{j,1} - TD_{j,r} \leq (1 + \frac{\delta}{100}) * Time_{j,r-1}$;

/Crossing Constraints $\forall i = 1..n \forall j = 1..m \forall k = 1..r$

(6) $TA_{i,k+1} < TD_{j,k+1} \vee TA_{j,k} < TD_{i,k}$;

/Overtaking Constraints $\forall i = 1..n \forall j = 1..m \forall k = 1..r$

(7) $TD_{i,k} < TD_{j,k} \rightarrow TA_{i,k+1} < TA_{j,k+1}$;

/Expedition time Constraints $\forall i = 1..n \forall j = 1..m \forall k = 1..r$

(8) $TA_{j,k} + ET_j \leq TD_{i,k} \vee TA_{i,k} + ET_i \leq TD_{j,k}$;

/Reception time Constraints $\forall i = 1..n \forall j = 1..m \forall k = 1..r$

(9) $TA_{i,k} + RT_i \leq TA_{j,k} \vee TA_{j,k} + RT_j \leq TA_{i,k}$;

Figure 2: Formal Model of the Railway Scheduling Problem.

Partition Proposals

Due to specific properties in the railway scheduling problem, several models can be adopted to distribute the problem.

Partition Proposal 1

The first way to distribute the problem is carried out by means of a graph partitioning software called METIS (METIS), for the purpose of this distribution, the model constraints are converted into binary constraints, this is trivial and the result is a binary CSP. METIS provides two programs *pmetis* and *kmetis* for partitioning an unstructured graph into k equal size parts. In this way, the railway

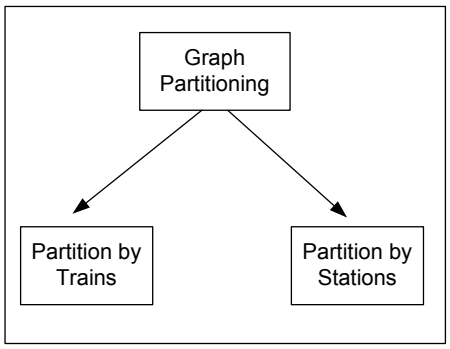


Figure 3: Distributed models: From Graph Partitioning to Train Partitioning and Station Partitioning.

scheduling problem can be modelled as a constraint network. This network can be partitioned in semi-independent subproblems by means of METIS. However, this software does not take into account additional information about the railway infrastructure or the type of trains to guide the partition, so the generated clusters may not be the most appropriate and the results are not appropriate. To improve the partition procedure, we extract additional information from the railway topology to obtain better partitions such as partition proposal 2 and 3.

Partition Proposal 2

The second model is based on distributing the original railway problem by means of train type. Each agent is committed to assign values to variables regarding a train or trains to minimize the journey travel. Depending on the selected number of partitions, each agent will manage one o more trains. Figure 4 shows a running map with 20 partition, each agent manages one train. This partition model has two important advantages: Firstly, this model allow us to improve privacy. Currently, due to the policy of deregulation in the European railways, trains from different operators work in the same railway infrastructure. In this way, the partition model gives us the possibility of partition the problem such as each agent is committed to a operator. Thus, different operators maintain privacy about strategic data. Secondly, this model allow us to manage efficiently priorities between different types of trains (regional trains, high speed trains, freight trains). In this way, agents committed to priority trains (high speed trains) will firstly carry out value assignment to variables, in order to achieve better journey travels

Partition Proposal 3

The third model is based on distributing the original railway problem by means of contiguous stations. Due to deregulation of European railways operators, long journeys may be scheduled. However, long journeys involve large number of stations at different countries with different railway policies. Therefore, a logical partition of the railway network can be carried out by means on regions (contiguous stations). To carry out this type of partition, it is important to analyze the

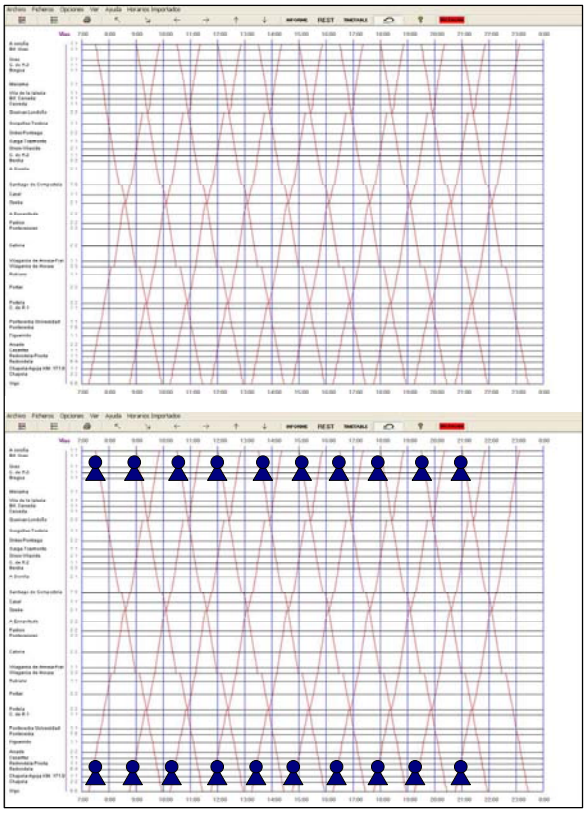


Figure 4: Distributed Railway Scheduling Problem. Proposal 2.

railway infrastructure and detect restricted regions (bottlenecks). To balance the problem, each agent is committed to a different number of stations. An agent can manage many stations if they are not restricted stations, whereas an agent can manage only few stations if they are bottlenecks. Furthermore, the agents committed with bottleneck have preferences to assign values to variables due to their domains are reduced (variable ordering).

Thus, the running map to be scheduled between two cities is decomposed in several and shorter running maps. Figure 5 (up) shows a running map to be scheduled. The set of stations will be partitioned in block of contiguous stations and a set of agents will coordinate to achieve a global solution (Figure 5 (down)). Thus, we can obtain important results such as railway capacity, consistent timetable, etc.

The Distributed Model

In the specialized literature, there are many works about distributed CSPs. In (Yokoo & Hirayama 2000), Yokoo et al. present a formalization and algorithms for solving distributed CSPs. These algorithms can be classified as either distributed stochastic search methods, synchronous backtracking or asynchronous backtracking (Yokoo & Hirayama 2000).

Our model can be considered as a synchronous model. It is meant to be a framework for interacting agents to achieve

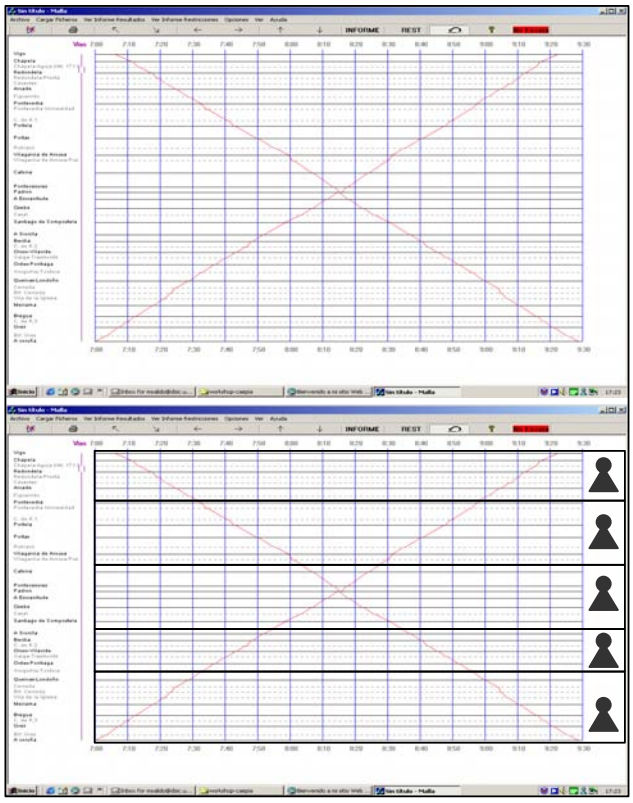


Figure 5: Distributed Railway Scheduling Problem. Proposal 3.

a consistent state. The main idea of our multi-agent model is based on (Salido, Giret, & Barber 2003) but partitioning the problem in k subproblems as independent as possible, classifying the subproblems in the appropriate order and solving them concurrently.

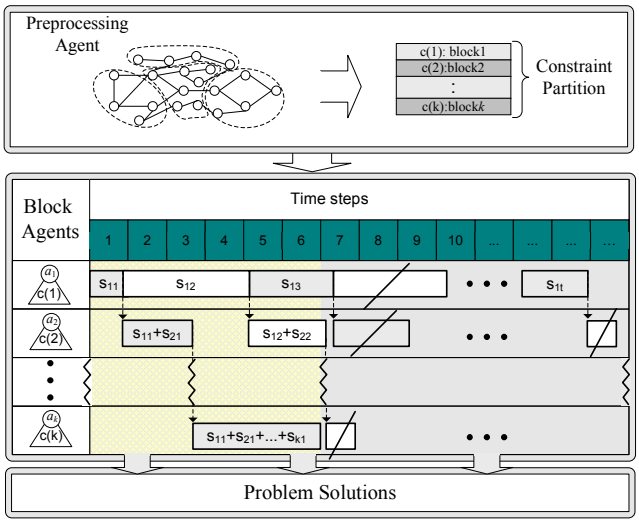


Figure 6: Multi-agent model.

In all our proposals, the problem is partitioned in k blocks or clusters in order to be studied by agents called *block agents*. Furthermore, a partition agent is committed to classify the subproblems in the appropriate order depending on the selected proposal. For instance, if Metis is selected to partition the problem, the partition agent must classify the subproblems such as the most interrelated problem is studied first.

Once the constraints are divided into k blocks by a *preprocessing agent*, a group of *block agents* concurrently manages each block of constraints. Each *block agent* is in charge of solving its own subproblem by means of a search algorithm. Each *block agent* is free to select any algorithm to find a consistent partial state. It can select a local search algorithm, a backtracking-based algorithm, or any other, depending on the problem topology. In any case, each *block agent* is committed to finding a solution to its particular subproblem. This subproblem is composed by its CSP subject to the variable assignment generated by the previous *block agents*. Thus, *block agent* 1 works on its group of constraints. If *block agent* 1 finds a solution to its subproblem, then it sends the consistent partial state to *block agent* 2, and both they work concurrently to solve their specific subproblems; *block agent* 1 tries to find other solution and *block agent* 2 tries to solve its subproblem knowing that its *common variables* have been assigned by *block agent* 1. Thus, *block agent* j, with the variable assignments generated by the previous *block agents*, works simultaneously with the previous *block agents*, and tries to find a more complete consistent state using a search algorithm. Finally, the last *block agent* k, working simultaneously with *block agents* $1, 2, ...(k-1)$, tries to find a consistent state in order to find a problem solution.

Figure 6 shows the multi-agent model, in which the *preprocessing agent* carries out the network partition and the *block agents* (a_i) are committed to concurrently finding partial problem solutions (s_{ij}). Each *block agent* sends the partial problem solutions to the following *block agent* until a problem solution is found (by the last *block agent*). For example, state: $s_{11} + s_{21} + ... + s_{k1}$ is a problem solution. The concurrence can be seen in Figure 6 in *Time step* 6 in which all *block agents* are concurrently working. Each *block agent* maintains the corresponding domains for its *new variables*. The *block agent* must assign values to its *new variables* so that the block of constraints is satisfied. When a *block agent* finds a value for each *new variable*, it then sends the consistent partial state to the next *block agent*. When the last *block agent* assigns values to its *new variables* satisfying its block of constraints, then a solution is found.

Evaluation

In this section, we carry out an evaluation between our distributed model and a centralized model. Furthermore, we evaluate the behavior of three proposed partition models. To this end, we have used a well-known CSP solver called CON'FLEX[1] which uses Forward Checking (FC) algorithm.

[1]It can be found in: http://www-bia.inra.fr/T/conflex/ Logiciels/adressesConflex.html.

Table 1: $< n, 20, 120 >$: 20 stations, 120 minutes frequency.

Trains (n)	*Variables*	*Constraints*
1	80	107
2	160	234
3	240	379
4	320	550
5	400	742
6	480	953
7	560	1186
8	640	1442
9	720	1717
10	800	2010

Table 2: $< 5, s, 120 >$: 5 trains, 120 minutes frequency.

Stations (s)	*Variables*	*Constraints*
10	200	527
20	400	742
30	600	1178
40	800	1608
50	1000	2073
60	1200	2555

This empirical evaluation was carried out over a real railway infrastructure that joins two important Spanish cities (La Coruna and Vigo). The journey between these two cities is currently divided by 40 stations. In our empirical evaluation, each set of random instances was defined by the 3-tuple $< n, s, f >$, where *n* was the number of periodic trains in each direction, *s* the number of stations and *f* the frequency. The problems were randomly generated by modifying these parameters.

Tables 1 and 2 show the parameters used to evaluate the behavior of the centralized model and the distributed model with the proposal partitions. We can observe that the complexity increased when the number of trains and stations increased. All instances maintain a frequency $f = 120$ minutes.

Table 1 shows the number of variables and the number of constraints generated when the number of trains in each direction increased from 1 to 10 in a railway infrastructure with 20 stations and a frequency of 120 minutes $< n, 20, 120 >$.

Table 2 shows the number of variables and the number of constraints generated when the number of stations increased from 10 to 60 in a running map with 5 train in each direction and a frequency of 120 minutes $< 5, s, 120 >$. Because the real railway infrastructure maintains 40 stations, we have virtually eliminated and added stations to carried out this evaluation.

General graph partitioning applications work well in general graphs. However, in the railway scheduling problem, we did not obtain good results using these softwares. We evaluate the partition proposal 1 by using METIS in several instances of Table 1. However, the obtained results were even worse in the distributed model than in the centralized model. We studied the partitions generated by METIS and we observed that the journey of a train is partitioned in several clusters, and each cluster was composed by tracks of trains in opposite directions. This cluster is easy to solve but very difficult to propagate to other agents. Furthermore, the following partition proposals make the contrary, that is, they never join tracks of trains in opposite directions.

So, we can conclude that the problem is very dependent of the partition that we carry out, and a general partition based on low connectivity is not always the best solution.

Figure 7 shows the running time of the instances presented in Table 1 meanwhile Figure 8 shows the running time of the instances presented in Table 2. In both Figures, the partition model selected was partition proposal 2, where the number of partition/agents was equal to the number of trains. In both figures, we can observe that the running time increased when the number of trains increased (Figure 7) and when the number of stations increased (Figure 8). However, in both cases, the distributed model maintained better behavior than the centralized model.

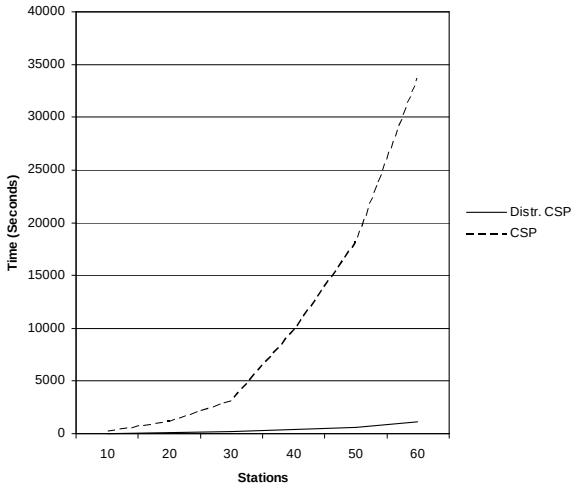


Figure 7: Running Time when the number of trains increased.

The partition proposal 2 was the best of the partition proposals, where we can schedule many trains in large railway infrastructure. However, how many partitions must divide the railway problem? If we select a large number of partitions, each subproblem is very easy, but the efficiency decreased due to communication messages. If we select a low number of partitions, each subproblem may be also difficult to solve. So, an appropriate number of partitions must be studied to solve the problem efficiently.

Figure 9 shows the running time with different partitions in problems where we fix the frequency (120 minutes), the number of stations (20) and the number of trains in each direction was increased from 5 to 10. Each instance was solved by the distributed model with different number of partitions (8,10,12,14,16,18,20 partitions). We can observe

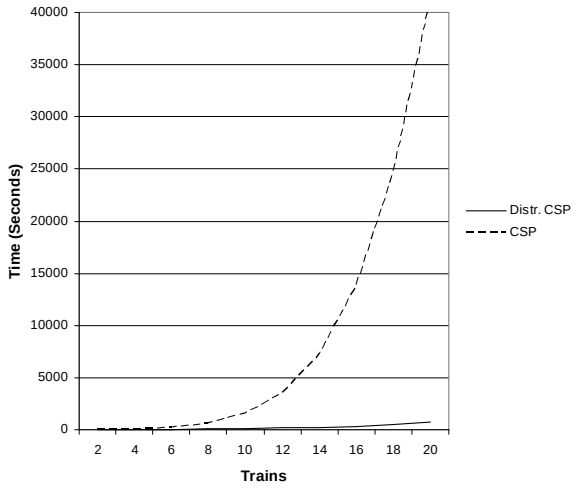


Figure 8: Running Time when the number of station increased.

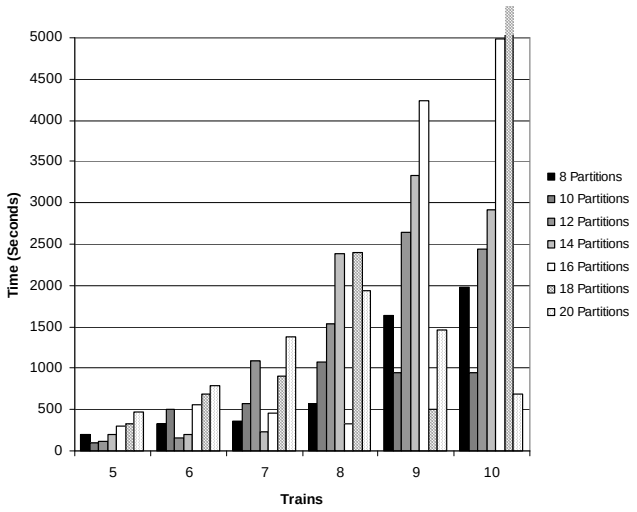


Figure 9: Running Time with different partitions.

the direct relation between the number of trains and the number of partitions. Thus, a agent was committed to schedule a train. If a agent was committed to schedule several trains, the efficiency decreased. Similar results happened when a train was scheduled by several agents.

The partition proposal 3, based on distributing the railway problem by means of contiguous stations was evaluated using the instances presented in Table 3.

This table shows the number of variables and the number of constraints generated when the number of trains in each direction increased from 1 to 10 in a railway infrastructure with 10 stations and a frequency of 120 minutes $< n, 10, 120 >$.

Figure 10 shows the running time of the centralized model and the distributed model, of the instances presented in Table 3 with a fixed number of partitions (6 partitions). It can be observed that the distributed model maintained a better behavior than the centralized model in all instances even with

Table 3: $< n, 10, 120 >$: 10 stations, 120 minutes frequency.

Trains (n)	*Variables*	*Constraints*
1	40	78
2	80	157
3	120	241
4	160	378
5	200	527
6	240	675
7	280	859
8	320	1078
9	360	1288
10	400	1515

a fixed number of partitions. However, in this type of distribution proposal, determining the appropriate number of partitions is difficult. It depends on the number of stations, the distance between them, the inclination of tracks, the number of tracks between stations, etc.

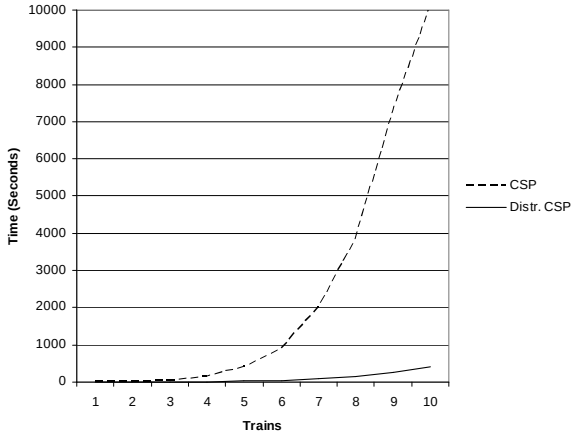


Figure 10: Running Time in problem with 6 partitions.

Conclusion and Future work

In this paper, we present a distributed model for solving the railway scheduling problem, in which several proposals are developed to distribute the railway scheduling into a set of sub-problems as independent as possible. Then, a set of *block agents* are incrementally and concurrently committed to building partial solutions until a global solution is found. The evaluation section shows the railway scheduling problem can be solved more efficiently in a distributed way. We are working on developing new heuristics to solve the distributed model in a more efficient way. Furthermore, it is necessary to built up a formal relation between the railway topology and the appropriate number of partitions.

References

Cordeau, J.; Toth, P.; and Vigo, D. 1998. A survey of optimization models for train routing and scheduling. *Transportation Science* 32:380–446.

Karypis, G., and Kumar, V. 1995. Using METIS and parMETIS.

Karypis, G., and Kumar, V. 1998. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing* 71–95.

Lova, A.; Tormos, P.; Barber, F.; Ingolotti, L.; Salido, M.; and Abril, M. 2006. Intelligent train scheduling on a high-loaded railway network. *Lecture Notes in Computer Science, (to appear)*.

METIS. http://www-users.cs.umn.edu/ karypis/metis/ index.html.

Nachtigall, K., and Voget, S. 1997. Minimizing waiting times in integrated fixed interval timetables by upgrading railway tracks. *European Journal of Operational Research* 103:610–627.

Salido, M.; Giret, A.; and Barber, F. 2003. Distributing Constraints by Sampling in Non-Binary CSPs. *In IJCAI Workshop on Distributing Constraint Reasoning* 79–87.

Schloegel, K.; Karypis, G.; and Kumar, V. 2003. Graph partitioning for high-performance scientific simulations. *Sourcebook of parallel computing* 491–541.

Schrijver, A., and Steenbeek, A. 1994. Timetable construction for railned. *Technical Report, CWI, Amsterdam, The Netherlands*.

Serafini, P., and Ukovich, W. 1989. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics* 550–581.

Silva de Oliveira, E. 2001. Solving single-track railway scheduling problem using constraint programming. *Phd Thesis. Univ. of Leeds, School of Computing*.

Walker, C., S. J., and Ryan, D. 2005. Simultaneous disruption recovery of a train timetable and crew roster in real time. *Comput. Oper. Res* 2077–2094.

Wooldridge, M., and Jennings, R. 1995. Agent theories, arquitectures, and lenguajes: a survey. *Intelligent Agents* 1–22.

Yokoo, M., and Hirayama, K. 2000. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems* 3:185–207.

Reverse Combinatorial Auctions for Resource Allocation in the Rescue Scenario

Silvia Suárez and Beatriz López
University of Girona, Spain
{sasuarez, blopez}@eia.udg.es

Abstract

In a disaster environment no single agent knows the disaster situation completely. So, rescue agents share their information in order to identify rescue tasks that are then allocated. In this paper we model the allocation problem as a reverse combinatorial auction and we apply some existing algorithms to solve it. Our results have been compared with the RoboCup classification of 2005.

Keywords
Resource Allocation, Combinatorial Auctions.

Introduction

Scheduling concerns the allocation of limited resources to tasks over time. It is a difficult problem that often requires ad hoc solutions. This is the case of the rescue scenario of the RoboCup rescue project [1]. In a simulated environment of an earthquake in a city and its first 300 cycles of simulation, rescue agents and victim agents interact. As in a real scenario, no agent has complete knowledge about the situation of the disaster. Coordination is required to share information about the situation and actuate consistently, so rescue agents collaborate to diminish the consequences of the disaster.

Rescue agents gather information on the area where they are placed, and report it to a central agent (station or office). This information includes victim localization, blocked roads, and fires. Next, the central agent facilitates information about some rescue tasks to an idle agent.

When deciding which resource agent should be assigned to a rescue task, a central agent has partial knowledge of the current situations of the agents. All the information concerning the agents' situations must be gathered in communication messages that require some time (simulation cycles in the RoboCup simulator). Rescue agents, conversely, know which actions are required to tackle a given task by considering several factors such as: blocked roads, other distended victims close to them, and their own survival rooms (since they can also be damaged), among others. It therefore seems appropriate that rescue agents are able to express their task preferences before central agents allocate them.

Auctions provide such a framework: agents make bids that represent their task performance preferences. A central agent, using the knowledge of the bids, can then make a final decision. In the rescue domain, in particular, when different tasks must be tackled at the same time, the appropriate framework is a combinatorial auction. In this paper we present the application of combinatorial auctions to coordinate the rescue agents in the RoboCup environment following the methods described in [5] and [6].

Rescue Domain Description

Disaster rescue is a complex domain in which it is hard to conduct real life experimentation. For this reason, we have used the simulator provided for the RoboCup [1] to simulate a disaster environment caused by an earthquake. In this simulated scenario, there are collapsed buildings, fires, blocked highways, people in a state of panic looking for a safe place and rescue agents such as fire brigades, ambulance teams, police forces and rescue headquarters. These last ones have to coordinate all kinds of rescue agents.

There are several communication and capability constraints in this scenario [3]. Regarding communication, for example, ambulance agents cannot detect victims and spend too much time looking for them. In this sense, effective communication among heterogeneous rescue agents about the position of victims is crucial to rescuing them.

With regard to planning and scheduling, agents have to make decisions about task execution. These decisions are related to which victim (in the case of an ambulance team) agents have to rescue first in order to maximize benefits from the system, such as reaching and rescuing the highest number of victims while minimizing damage in the surrounding area. In the case of police forces, for example, they need to determine which roads they should unblock first, taking into account that other kinds of rescue agents can be stopped by these blocked roads [2]. Besides, in the case of fire brigade agents, they first need to extinguish fires in buildings containing the highest number of potential victims.

Two main problems can be distinguished in this scenario: planning which tasks should be performed first (for example, a plan to rescue a victim should not be made before the road is unblocked) and scheduling which resources are used to perform the tasks. In previous works [2] we have studied temporal dependencies related to the planning problem. In this paper, we do not consider task dependencies and only deal with the allocation of a set of agents to rescue tasks.

Reverse Combinatorial Auction Formulation

A traditional reverse combinatorial auction is a type of combinatorial auction in which the buyer wants to obtain a set of items, M={1,2,…,m}, and the sellers submit a set of asks, B={B_1,B_2,…,B_n}. An ask is a tuple B_j=(S_j,a_j), where $S_j \subseteq M$ is a set of items and a_j, $a_j \geq 0$ is an asking cost. How to determine the combination of winner bids (i.e. the ones that minimize the cost) is known as the winner determination problem. The integer programming formulation for the winner determination problem consists of labeling the asks as winning or losing so as to minimize the buyer's cost under the constraint that the buyer obtains each item:

$$\text{Min} \sum_{j=1}^{n} a_j x_j \quad \text{subject to} \sum_{j|i \in S_j} x_j \geq 1 \quad i = 1,2,...,m \qquad (1)$$

The X_j variable can take 0 or 1 value and there is one for each bid B_j. It means that if X_j=1, B_j is accepted. The constraint ensures that at least one bid is allocated for each item.

Rescue Formulation

In the rescue domain, the problem of assigning rescue agents to rescue tasks can be formulated as a reverse combinatorial auction problem. To do so, we consider that central agents are buyers, and rescue agents are sellers who are submitting asks for rescue tasks. We then define the following 4-step communication protocol:

1. The rescue agents send the local tasks they can detect in their surroundings.
2. The central agents receive the tasks from the rescue agents and return the complete list of tasks to each agent.
3. With information about all the tasks, the rescue agents send bids corresponding to combinations of tasks to the central agents.
4. The central agents determine the winners, using the winner determination algorithm for combinatorial auctions, and send the results to the rescue agents.

Since each message can be sent/received in a simulation cycle, the protocol requires four cycles. Consistently, a combinatorial auction is performed once every 4 simulation cycles.

Bid Generation

The rescue agents generate bids corresponding to combinations of tasks to be performed in sequential order. Bids consist of a list of tasks and the costs assumed by the agents to carry out these tasks. The cost is determined by the sum of distances from the agents to the tasks.

Solving the Winner Determination Problem

Winner determination algorithms are aimed at solving the optimization problem defined by a combinatorial auction as explained in the previous section. For our particular problem, we have used a formulation based on [5]. It is an A* search algorithm where the bids are organized in a structure called a bidtree. The bidtree is used to define a heuristic to order the bids in the A* search according to [6].

The Bidtree Formulation

A bidtree is a binary tree whose principal purpose is to support content-based lookup of bids [6]. The tree depth is m+1, where m is the number of tasks in the problem. The bids appear in the leaf nodes. Each level of the bidtree represents one task and the in and out branches show if the task belongs or not to the bid. For example, suppose we have some victims at some simulation time with the following identifications, Id_1, Id_2, Id_3, Id_4, and assume that our ambulance central has received the following bids, B_1(s_1, 30), B_2(s_2, 27), B_3(s_3, 17), B_4(s_4, 32), B_5(s_5, 39), B_6(s_6, 14), B_7(s_7, 12), B_8(s_8, 65), B_9(s_9, 24), B_{10}(s_{10}, 33), with the corresponding set of tasks as follows:

s_i= (s_1=[Id_2, Id_4], s_2= [Id1, Id_2], s_3=[Id_1], s_4=[Id_2, Id_4], s_5=[Id_1, Id_2, Id_3], s_6=[Id_4], s_7=[Id_1], s_8=[Id_1, Id_2, Id_3, Id_4], s_9=[id_1, id_4], s_{10}=[id_3, id_4]).

Taking into account that bid B3 has the same task set as bid B7 and that the cost of B3 is higher than the cost of B7, B3 can be erased from the bid set as it will never generate an optimal solution. The same is true for bid B4 with regard to bid B1.

Bid ordering

Taking into account how many times the tasks appear in the bids (bid count), the tasks can be ordered with either an increasing or a decreasing bid count. Using the increasing bid count, the order of tasks for the bidtree is:

id_1, id_4, id_2, id_3. The bidtree with this sorting is shown in Figure 1.

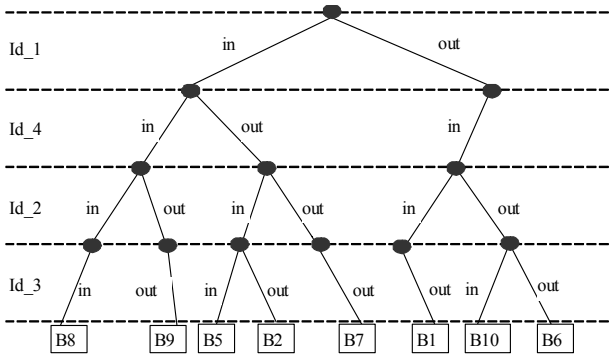


Figure 1. Bidtree with increasing bid count

On the other hand, taking into account a decreasing bid count, the order that we get is: id_3, id_2, id_1, id_4. The bidtree for this case is presented in Figure 2.

.

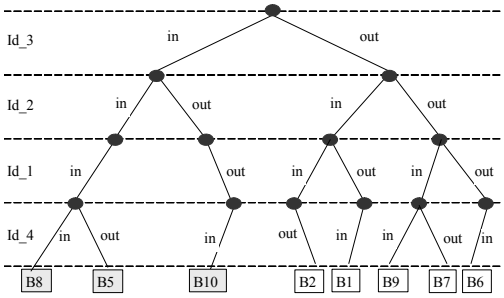


Figure 2. Bidtree with decreasing bid count

Generating the Search Tree

The bidtree is used to generate the search tree. The algorithm based on A* for generating the search tree is presented below.

Let B = {B_1, B_2,…B_n} be the set of bids submitted to be considered in the winner determination process. Let Bc={Bc_1, Bc_2,…Bc_m} be the subset of bids of B that do not include any tasks that have already been allocated and which are still available to be appended to the search path. Let Bl={Bl_1, Bl_2,…Bl_n} be the subset of bids in the leave nodes of the left subtree of the bidtree (in branch). These are the bids in shaded squares in Figure 2. Let Br= {Br_1, Br_2,…Br_n } be the subset of bids in the leave nodes of the right subtree of the bidtree (out branch). These are the bids in unshaded squares in Figure 2. Let $alloc_{best}$ be a subset of B which is the best allocation found so far (i.e. with the minimum cost), and C($alloc_{best}$) be the total cost of $alloc_{best}$. Let Bo={Bo_1, Bo_2,…Bo_n } be the set resulting from Bc ∩ Br. Analogously, let Bq(B_i)= {Bq_1,Bq_2,…Bq_n} be a set which stores the bids that conflict with bid Bi. In addition, for one partial allocation *alloc*={B_1…B_n}, it is possible to define:

$$Bq(alloc)= \bigcup_i B_i \qquad (2).$$

The steps of the algorithm are shown in Figure 3. This algorithm is exponential in the worst case, when b→ ∞ (b=number of bids), and finds the optimal solution, always the least costly path in the search tree, if any exists.

Set C($alloc_{best}$) = {}

For each b from Bl

1. b=one bid from set Bl.
2. *alloc*={}
3. Bc(*alloc*)= B – Bq(*alloc*) – (*alloc*) (3)
4. Bo=Bc(*alloc*)∩Br
5. if Bo != {} then

   b_j= one bid from Bo which has the lowest cost.

   If b_j then

   alloc=*alloc* ∪ bj
6. if (|tasks(*alloc*)| = total) then

   if C(*alloc*) < C($alloc_{best}$) then

   C($alloc_{best}$) = C(*alloc*)

   Return to point 1, next b.

   else

   Backtracking
7. Return to point 3.

Figure 3. Algorithm to generate the search tree for the winner determination in combinatorial auctions based on A*

Regarding the decreasing order of our example (Figure 2), the Bl set is {B8, B5, B10}, and the Br set is {B2, B1, B9, B7, B6}. First, *alloc* B8 is explored. Since B8 contains all tasks, no further exploration is carried out. So, B8 is the first feasible solution found for the algorithm. Next B5 is explored. Regarding the Bc set, for instance, the B and Bq sets for *alloc* = B5 are as follows: B={B1, B2, B5, B6, B7, B8, B9, B10}, Bq(B5)= {B1, B2, B7, B8, B9, B10}. And according to step 3: Bc($alloc_{B5}$)= B-Bq(B5)-B5, Bc($alloc_{B5}$)= B6, and Bo=Bc ∩ Br, so Bo=B6. As Bo is composed just for one bid, no bid ordering which takes cost into account is required. Since {B5, B6} is a combination that contains all the tasks, it is another feasible solution. Finally, the total search space generated is presented in Figure 4. The best

allocation found $alloc_{best}$ is {B5,B6}. And the cost $C(alloc_{best})= 55$.

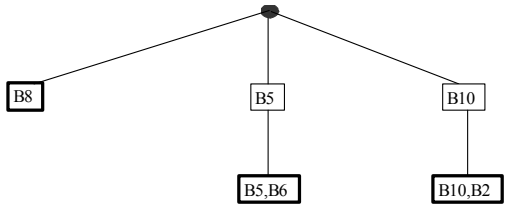


Figure 4. Search tree. Solutions provided in bold squares.

With the increasing bid ordering (Figure 1), the solution of the problem is the same. However, the number of nodes explored is higher. This is something that has been already studied in [6]. The ideal task ordering is one in which the left subtree (the in branch) is always maximally larger than the right subtree (the out branch).

Regarding our example, the ambulance agents who have submitted bids B5 and B6 are the winners and can carry out the set of tasks in their correspondent bids.

Results

We performed 10 simulations and we recorded the results according to the V score defined in the competition [1], which is the following:

$$V=(P + S/Sint) * sqrt(B/Bint) \qquad (4).$$

The higher V value for a map, the better the rescue operation (maximum V value is 97).

Our V average is 62.73. As a frame of reference, the scores of the first four teams in the final of the latest RoboCup Rescue competition (2005) for the Kobe scenario map were 83, 79, 69 and 58. So, our agents have obtained a good score in this frame. Note that in our experiments we have not used any planning method to take into account either task precedence (only re-scheduling as explained in [2]) or path planning computation for rescue agents. Therefore, we believe that our results are quite encouraging. If we include planning competences in our agents in the future, we believe that our results will be even more satisfactory.

Conclusions and Future Work

This paper presents the application of reverse combinatorial auctions for task scheduling in rescue operations following [5] and [6]. The RoboCup rescue simulator has been used as a test bed. The allocation process is repeated with every 4 cycles of simulation.

As a future work, we are thinking about introducing the expected utility theory [7] to improve the planning process in the ambulance team agents. In this sense, the utility attained by agents from scheduling victim rescues is maximized. We are also thinking of applying our approach to one decentralized scheduling model.

Acknowledgements

This work was supported by the Spanish MEC project TIN 2004-06354-CO2-02 and the DURSI-AGAUR 00296 SGR (2005).

References

[1] RoboCupRescue Simulator, http://www.robocup2005.org/roborescue/

[2] Suárez S., Collins J., López B. Improving Rescue Operation in Disasters. Approaches about Task Allocation and Re-scheduling. In Proceedings of PLANSIG 2005. London, UK, 2005.

[3] Takeshi M. How to develop a RoboCup rescue agent. http://ne.cs.uec.ac.jp/~morimoto/rescue/manual/

[4] Sandholm, T., Suri, S., Gilpin, A., and Levine, D. 2005. CABOB: A Fast Optimal Algorithm for Winner Determination in Combinatorial Auctions. Management Science 51(3), 374-390, special issue on Electronic Markets.

[5] Collins, J. "Solving Combinatorial Auctions with Temporal Constraints in Economic Agents." PhD thesis, University of Minnesota, June 2002.

[6] Collins J., Demir G., and Gini M. "Bidtree ordering in IDA* combinatorial auction winner-determination with side constraints." In Lecture Notes of Artificial Intelligence I2531, pages 17-33. Springer-Verlag, 2002.

[7] A. Babanov, J. Collins, and M. Gini. "Harnessing the search for rational bid schedules with stochastic search and domain-specific heuristics." In Proc. of the Third Int'l Conf. on Autonomous Agents and Multi-Agent Systems, submitted, New York, July 2004. AAAI Press.

The Aspect of Quality in Scheduling Painting Tasks for an Automatic Robotic Spray System

Ewa Kołakowska and Ole Madsen

Department of Production
Aalborg University
Fibigerstræde 16,
DK-9220 Aalborg Øst,
Denmark
e-mail: ewa@iprod.aau.dk, i9om@iprod.aau.dk

Abstract

Paint planning is one of the key issues in fully automatic robotic spray painting systems. Generation of the precise spray painting paths is no longer sufficient to cope with high demands concerning optimization of both the lead-time and painting quality. In this context it is the sequence of the painting tasks that affects the efficiency of painting process. This paper presents a novel idea of improving the painting process by design of a scheduler that captures the aspect of quality. The proposed scheduler is based on the new concept of building a Product Process Structure Tree. It represents all possible solutions, constrained by quality requirements that are defined by process technical rules. By means of the Product Process Structure Tree the scheduling problem can be formulated as the constraint satisfaction problem. It is the first time the constrain satisfaction techniques are applied in the field of automatic spray painting.

Introduction

The automation of the industrial painting and inculcating a teachingless programming to the spray painting robot has been the subject of intensive research for over 10 years. Commonly used methodology aims at automatic generation of the trajectory of the spray-gun path and transferring the path to the robot motions. It originates from [1], where the first system architecture of an automatic robotic spray painting was introduced. The architecture consists of four general subsystems: reading part data, paint planning, motion planning and generation of the robot program. The verification tests showed that automation of the robotic painting reduces the programming time of a robot by 50%. Though, it was still not possible to apply the method in the industry mainly due to small number of elementary geometries considered and assumption that there is CAD data available. The approach was later undertaken as a part of a research project *FlexPaint* [2]. The system was extended by the laser triangular sensor, which enables to scan the part to obtain 3D measurement of point data of the object and therefore determine part geometry without prior 3D CAD data. The features of the paint planner were extended to capture more complex geometries, such as cavities or ribs. Finally, a collision avoidance system was designed and integrated with the motion planner. However, all the improvements did not assure satisfactory results in terms of painting quality. In *FlexPaint* the spray painting path was represented by the set of paint strokes. The sequence of paint strokes was planned with the objective of minimizing the Cartesian distance between the strokes. This method did not always provide acceptable quality of painting.

As a consequence in this paper the hypothesis is posed, that changing the order of paint strokes can significantly improve the painting quality. The designed paint strokes scheduler is the first attempt to formalize the way of coping with combined quality and management issues in the automatic spray painting. The inspiration for this research was the work presented in [3], which shows a general approach, *allocation-state space expansion algorithm* (ASSE), for allocation of the tasks to the resources. The approach captures the data structure of all possible solutions. The quality is an extra parameter to be optimized, which is first considered while searching the data structure for an optimal sequence. However, as it is stated in [4], case studies on using the ASSE algorithm for sequencing painting tasks shows that the scheduler becomes a bottleneck if the number of tasks exceeds 5 paint strokes, while it is common for complex parts to have up to 500 tasks ascribed to a single robot. This paper undertakes the problem and describes it as a constraint satisfaction problem. The study presented here is limited to single resource, multiple tasks environment. The achievement of this work is the improvement of the painting quality by designing the Product Process Structure algorithm, which captures several quality rules and enables scheduling 500 paint strokes. This paper also shows the new application for the constraint satisfaction techniques, which is in the field of automatic industrial painting.

Design of the System Architecture

The scheduler discussed in this paper is a part of automatic robotic spray painting system, which consists of four main entities: object geometry recognition, paint path planner, motion planner and scheduler. They are integrated as shown in figure 1.

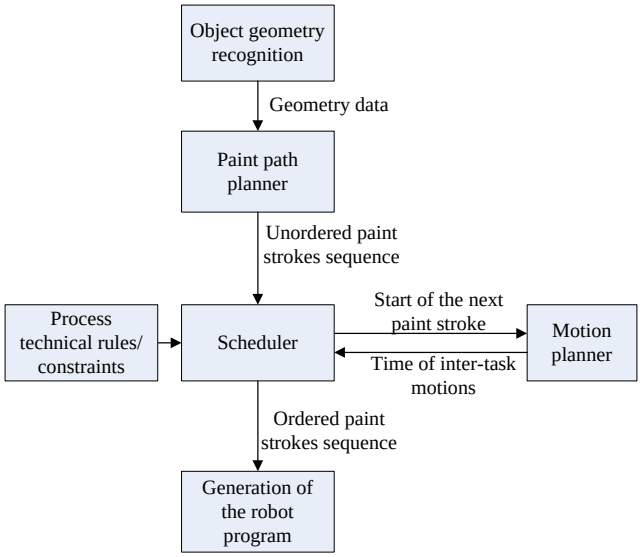


Fig1. System architecture of the robotic spray painting system

The *object geometry recognition* returns the geometry data of a part. This information is then passed to the *paint planner*, which generates the paint path represented by a list of paint strokes. The simplified visualization of the paint planner output is depicted in figure 2.

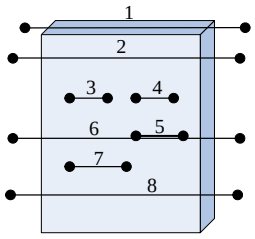


Fig2. Example of spray paint path

The paint strokes are *scheduled* using interactive communication with the motion planner while the search space is limited by the process technical rules and equipment constrains, such as robot limitations and size of a paint cell.

For most of the painting guns a paint stroke does not have a uniform distribution of paint within the cross section. Due to this property all of the paint strokes are planned in a way that their outermost parts are overlapping. Additionally, some of the paint strokes can be completely covered by another paint stroke, as it is the case for paint strokes number 5 and number 6 in figure 2. Creating multiple layers of paint is a necessity when the required paint thickness is too large to be performed within a single paint stroke. The overlaps create dependencies between paint strokes, which should be considered in the design of the scheduler. The influence of the overlaps on the paint quality is discussed in the following section.

Concept of Painting Quality in Terms of Scheduling

According to the international painting standards [5] and [6] the *painting quality* can be defined by a number of parameters such as: color, gloss, environmental impact, dry paint thickness and hardness of the paint film. It can be adjusted by means of the paint, painting equipment and painting techniques. Moreover, in the automatic painting it can be influenced by the precision of the paint path planning and the robot motion planning. In this paper it is claimed that the quality can also be changed by scheduling the sequence of painting tasks, which are given by the paint path planner.

In this context the *quality* is defined by means of the dry paint thickness covering the surface of a part. It can be checked in visual inspection and measured by several types of equipment and methods, e.g. eddy current thickness gages, magnetic thickness gages and the penetration method, thoroughly described in [7]. The term *high quality* would therefore describe the acceptable deviation of dry paint thickness. The term *low quality* would be ascribed to the parts that encounter defects. The most common painting defects, which can be reduced by terms of the scheduler, are:

- Overpaint and underpaint - occurs when the thickness of the paint covering a part is under or over a certain limit specified for each part.
- Paint dust - describes not smooth surface of the painted part. It appears when the paint is sprayed on the top of the dry paint.
- Poor adhesion, blistering and cracking – they reveal as dips, flows and rollers of paint.
- Sliding paint – results from spraying too much paint on the wet surface of the painted part. It appears as swellings of the paint.
- Runners – defined as the drops of paint, congealed on the edges of the part. They are due to the effluent paint.

There are several paint stroke features that can decrease the chance for the defects to appear and therefore influence the painting quality. Some of them will be exemplified in this paper.

Length of a paint stroke: It is common that there are paint strokes with many different sizes within each paint path. As it was verified in [4] short paint strokes should be

sequenced before the overlapping long ones, otherwise there may occur painting defects such as sliding paint, dips, flows and rollers. The length of the paint strokes can be represented by the Cartesian distance from the start to the end position of the paint stroke. For clarity of this paper all the paint strokes have one of two available states: either they are short or long. Their length is identified with respect to the average length of paint strokes, which is calculated for each part individually. Crossing the average length in positive direction defines a long paint stroke, crossing the average length in negative direction defines a short paint stroke. For the painting path exemplified in figure 2 the paint strokes number 3, 4, 5, 7 should be scheduled before the paint strokes number 1, 2, 6, 8.

Position of a paint stroke: Position of a paint stroke relates to the gravity. It is described in the World Reference Frame, with respect to a part. Generally it is received that parts should be painted from the top to the bottom, because due to the force of gravity the paint dust falls down and sticks to the surfaces below. This painting direction assures also better flow of the paint. According to this rule the paint stroke number 1 from figure 2 will be sequenced first and the paint stroke number 8 will be sequenced last.

Shape of surface: In the present system each paint stroke carries the information about the surface of the part it is ascribed to, that is flat, curved or cavity surface. It is recommended that the paint strokes with the flat features are performed after the paint strokes with the curved and cavity features, since the paint-dust from the later strokes is the most visible on the flat surface.

Recapitulating, the following process technical rules for scheduler can be defined:

- **a)** Short paint stroke should be scheduled before an overlapping long one.
- **b)** Paint stroke from the top should be scheduled before the one from the bottom.
- **c)** Paint stroke with the flat feature should be scheduled after the paint stroke with the curved or cavity feature.

Grouping paint strokes

The process technical rules are used here as constrains for limiting search space of tasks to be scheduled. They apply to all paint strokes and therefore require that the paint strokes carry the same amount of information about their features, e.g. length, position and shape of a surface. The paint strokes that have all the same states of the features are said to belong to the same group. Figure 3 shows an example with four groups.

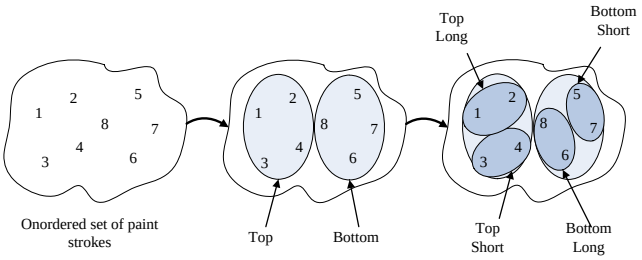


Fig3. Example of grouping paint strokes

Executing all paint strokes from each group results in a certain *product state*. All product states need to be achieved in order to reach the completion state, which is the state in which all the painting tasks are finished.

Grouping is performed in a certain order of paint strokes features. The paint strokes, which features have the biggest impact on the quality, are grouped first. Figure 3 is an example where the position of a paint stroke is more crucial in terms of quality than the length of a paint stroke.

Design of the Product Process Structure Tree

The quality is captured in the design of the scheduler by identifying the permissible product states. Product states can be interpreted as the stages of the process that have a certain effect on the product. They carry information about the tasks that need to be processed but do not determine their sequence. Product states are defined based on process technical rules that are important for the quality and correspond to appropriate groups of paint strokes. The quality of painting depends on the product states' sequence, which is called *Product Process Structure* (PPS). The product states are ordered according to the weights of the groups, w_g. The highest weights are given to the groups that have the biggest influence on the quality. Weighting of groups is exemplified in table 1.

Table1. Weighting of groups

Groups	Weights
Top-Short	w_g=3
Top-Long	w_g=2
Bottom-Short	w_g=2
Bottom-Long	w_g=1

The product process structure can be presented using graph method, described in [8] and for the data presented in table 1 it is depicted in figure 4. It takes the shape of the tree with the root in the initial state of the product and the leaves leading to the completion state. The nodes represent the product states, out-going arrows point at states that must be carried out from a given product state and in-going arrows point at which processes need to be performed to reach a particular state. If some of the product states have

the same weight, w_g, they are connected with the arc and called the *parallel product states*. Changing their sequence does not have impact on the quality.

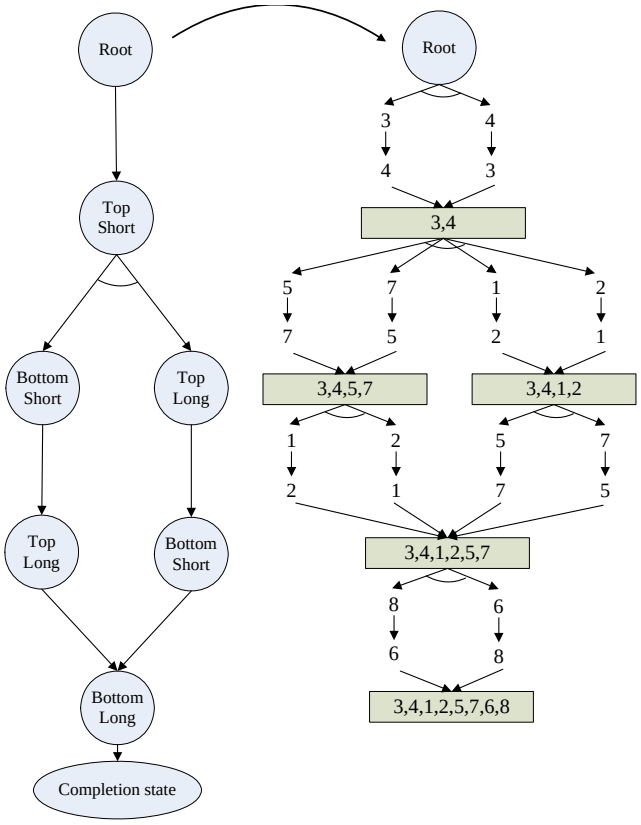


Fig4. Product process structure algorithm

Scheduling Algorithm

PPS algorithm is a systematic way of introducing quality rules. It shows the way through the product states that fulfils quality requirements and contains many possible ways of sequencing painting strokes. The final paint stroke order is found by performing a search through the PPS Tree with the objective of minimizing lead time while satisfying quality requirements. Scheduling of paint strokes can be done within constraint satisfaction framework. In this context all paint strokes are interpreted as the set of variables, each of which is associated with a domain. Domain consists of a set of numbers
{0, 1, 2, …, number_of_paint_strokes-1}, that correspond to the position in the ordered paint strokes sequence. PPS tree and lead time requirement restrict the values the paint strokes can take.

Conclusions

In this paper a painting process was examined not only with respect to the lead time, but also with respect to the painting quality. The quality aspect was captured by the process technical rules. They were presented using graphical method and took the form of a PPS tree with the permissible product states in each of the nodes. The PPS algorithm can be used as a constraint for scheduling and determine degrees of freedom for the scheduler. It can be viewed as the method for grouping the paint strokes. Applying PPS for scheduling reduces significantly the computational power and at the same time increases the painting quality.

In closure of this paper it should be mentioned that the designed scheduling algorithm has not been implemented yet. It is a part of a currently on-going European research program *Eco2Painter* [9] to test the proposed method. Its evaluation will be reported as the completion of this work.

Acknowledgements

This work is supported by the COOP project *Eco2Painter* (COOP-2005-017809).

References

[1] N. Asakawa, Y. Takeuchi, “Teachingless Spray-Painting of Sculptures by an Industrial Robot”, *Proc. of IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, 1997.

[2] A. Pichler, M. Vincze, H. Andersen, O. Madsen, K. Häusler, ”A Method for Automatic Spray Painting of Unknown Parts”, *Proc. of the 2002 IEEE International Conference on Robotic and Automation*, May 2002.

[3] H. Holm, “Information Structures for Design of Feedback Resource Distribution Control Systems for Discrete Part Manufacture”, *Proc. of 2nd International Workshop on Supply Chain Management and Information Systems*, July 2004.

[4] S. Nielsen, K. B. Hansen, M. S. Nielsen *”Konstruktion af tilstandsbaseret anlægsstyring til flexibelt maleanlæg”*, Master thesis (in Danish), Aalborg University, 2004.

[5] American Society for Testing and Materials (ASTM) Standards; Part 27, Paint – Tests for Formulated Products and Applied Coatings.

[6] Federal Test Method Standard No. 141a Paint, Varnish, Lacquer and Releted Material; Methods of Inspection, Sampling and Testing.

[7] ASM Handbook Committee, “Metals Handbook”, 9th Ed., vol.5.

[8] N. J. Nilsson, “Principles of Artificial Intelligence”, Springer Verlag, 1982.

[9] *Eco2Painter*, http://www.eco2painter.org