22nd International Conference on Automated Planning and Scheduling

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

COPLAS 2012

Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems

Edited by
Miguel A. Salido, Roman Barták

Editors

Miguel A. Salido, Universidad Politécnica de Valencia, Spain
msalido@dsic.upv.es

Miguel A. Salido is supported by the research project TIN2010-20976-C02-01 (Min. de Economia y Competitividad, Spain) and project PIRSES-GA-2011-294931 (FP7-PEOPLE-2011-IRSES)

Roman Barták, Charles University, Czech Republic
bartak@ktiml.mff.cuni.cz

Roman Barták is supported by the Czech Science Foundation under the contract P202/10/1188

Organization

Miguel A. Salido, Universidad Politécnica de Valencia, Spain
contact email: msalido@dsic.upv.es

Roman Barták, Charles University, Czech Republic
contact email: bartak@ktiml.mff.cuni.cz

Program Committee

Federico Barber, Universidad Politécnica de Valencia, Spain

Roman Barták, Charles University, The Czech Republic

Amedeo Cesta, ISTC-CNR, Italy

Minh Binh Do, NASA Ames Research Center, USA

Enrico Giunchiglia, Universita di Genova, Italy

Peter Jarvis, NASA Ames Research Center, USA

Eva Onaindía, Universidad Politécnica de Valencia, Spain

Nicola Policella, European Space Agency, Germany

Hana Rudová, Masaryk University, The Czech Republic

Francesca Rossi, University of Padova, Italy

Migual A. Salido, Universidad Politecnica Valencia, Spain

Pascal Van Hentenryck, Brown University, USA

Gérard Verfaillie, ONERA, Centre de Toulouse, France

Vincent Vidal, CRIL-IUT, France

Petr Vilim, ILOG, France

Toby Walsh, UNSW, Sydney and NICTA, Australia

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

Foreword

The areas of AI planning and scheduling have seen important advances thanks to the application of constraint satisfaction models and techniques. Especially solutions to many real-world problems need to integrate plan synthesis capabilities with resource allocation, which can be efficiently managed by using constraint satisfaction techniques. The workshop will aim at providing a forum for researchers in the field of Artificial Intelligence to discuss novel issues on planning, scheduling, constraint programming/constraint satisfaction problems (CSPs) and many other common areas that exist among them. On the whole, the workshop will mainly focus on managing complex problems where planning, scheduling and constraint satisfaction must be combined and/or interrelated, which entails an enormous potential for practical applications and future research.

Miguel A. Salido, Roman Barták
COPLAS 2012 Organizers
June 2012

Contents

SMT Spatio-Temporal Planning

Lamia Belouaer
GREYC, Université de Caen Basse Normandie
Boulevard du Maréchal Juin
BP 5186 14032 Caen Cedex, France

Frédéric Maris
IRIT, Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 9, France

Abstract

Solving real planning problems requires to consider spatial and temporal information. Indeed, to be solved more efficiently many real world problems need to take the action duration, the instants of effects occurrences, the instants of requiring preconditions... and the space in which the mission is accomplished by defining the different actions zones and know the path between these zones into account. In this paper we are interested in planning problems which consider spatial and temporal dimensions. To our knowledge there is no model expressing both spatial and temporal dimensions in planning problems. The main contribution of this paper is to present an approach allowing the resolution of spatio-temporal planning problems. For this purpose we define a new SMT (Sat Modulo Theory) encoding rules for spatio-temporal planning. This new code can compile and solve spatio-temporal planning problems for which all solutions require simultaneous actions in a 2D space.

Introduction

The classical planning framework allows only limited representation of real world aspects. We aim to expand this framework with spatial and temporal aspects while maintaining good performance. In planning applications in real-world these two dimensions are useful. Indeed, to be solved more efficiently many real of these problems need to take the action durations, the instants of effects occurrences, the instants of requiring preconditions... and the space in which the mission is accomplished into account by defining the different actions zones and the path between these zones. For instance, manipulating objects in a nuclear plant, the building evacuation in case of fire, the service in a restaurant... are some examples of such problems. To solve this kind of problem we aim at developing an approach based on spatial and temporal reasoning. However, to our knowledge there is no model expressing planning problem by considering both spatial and temporal dimensions at the same time and there is no planner solving such a problem.

The main contribution of this paper is to present an approach allowing the resolution of spatio-temporal planning problem. The idea is to allow synchronization of non-instantaneous actions in space. To this aim, we consider the encoding rules of TLP-GP-2 (Maris and Régnier 2008a) (Maris and Régnier 2008b) and we integrate new rules based on *SpaceOntology* (Belouaer, Bouzid, and Mouaddib 2010), and which can encode the spatial dimension of a planning problem.

The temporal planner TLP-GP-2 is based on the use of a simplified planning graph and a SAT Modulo Theory (SMT) solver. The problem representation language that TLP-GP can process allows a much greater flexibility than PDDL2.1 (Fox and Long 2003) for defining temporal domains and problems. Although its expressive power is identical to that of PDDL2.1, the extensions implemented in TLP-GP allow the user to express real-world problems much more easily (Cooper, Maris, and Regnier 2010a). To provide a rich temporal representation of actions, time points within actions can be used, other than start and end, along with sets of simple linear binary (in)equality constraints between time points. On the other hand, high-level modalities allow the user to define complex relationships between a condition or effect and an interval, including being valid over the whole interval, at some point in the interval, within a sub-interval or being subject to a continuous transition over the interval. TLP-GP can also take into account, in a very natural way, exogenous events as well as temporally extended goals. It is complete for the temporally expressive sublanguages of PDDL2.1 using the transformation method of (Cooper, Maris, and Regnier 2010b) to restore its completeness when handling temporally-cyclic problems.

SpaceOntology considers different space's representations: qualitative (topological relation and fuzzy distance), quantitative (numeric distance) and hierarchical. Also, *SpaceOntology* defines a set of rules to deduce new information to complete the description of the environment. This allows a complex spatial description and an easy management of different spatial aspects. In this paper, we consider four concepts defined in *SpaceOntology*. Namely (Belouaer, Bouzid, and Mouaddib 2010): spatial entity, numeric distance, fuzzy distance, hierarchical representation. This paper presents an extension of the temporal planner TLP-GP-2, called ST-SMTPLAN, to take into account temporal dimension with some spatial aspects of *SpaceOntology* (numeric distance, fuzzy distance and hierarchical link).

This paper is organized as follows. First, we present vari-

Supported by ANR Project ANR-10-BLAN-0210.

ous related works on spatial and/or temporal planning. Then, the preliminaries part presents a set of required notations and definitions. Next, we define a study case in order to illustrate our claim. After, we present some of the spatio-temporal encoding rules needed to solve the problem described relative to our study case. The discussion section shows theoretically the soundness and the completeness of this two-dimensional encoding. Finally, the conclusion presents the lines of research opened by this work.

Related Work

To solve real planning problems, one of the major challenges is to consider spatial and temporal dimensions. However, to our knowledge there is no model allowing us to represent a planning problem by considering both spatial and temporal knowledge at the same time and no planner allowing us to solve such a problem. This part is a literature review of various works on temporal planners and spatial planners.

Temporal Planners

Most of temporal planning systems use an additional constraints solver for task scheduling. In the past, many planners have used a hierarchical plan-space (HTN). They used a temporal logic based on instants and intervals, together with a Time Map Manager which manages the temporal constraints. This is the case for planners such as IxTeT (Ghallab and Alaoui 1989), (Laborie and Ghallab 1995) or HSTS (Muscettola 1993). When (Cushing, Kambhampati, and Mausam 2007) published their important work on temporally-expressive planning, the majority of efficient temporal planners developed between 1994 and 2007 were incapable of solving temporally-expressive problems, although some planners have since been improved to solve such problems. Apart from HTN-type planners which present a relatively poor performance, only few of them can solve this type of problem. Some of them such as CRIKEY3 (Coles et al. 2008), VHPOP (Younes and Simmons 2003), LPGP (Long and Fox 2003), TLP-GP-1 (Maris and Régnier 2008a) (Maris and Régnier 2008b), and the most recent version of LPG (Gerevini, Saetti, and Serina 2010), perform a search algorithm coupled with a Simple Temporal Network (STN) solver. Others such as TM-LPSAT (Shin and Davis 2004) , the planner of (Hu 2007), TLP-GP-2 (Maris and Régnier 2008a) (Maris and Régnier 2008b), STEP (Huang, Chen, and Zhang 2009), use a similar method to that used by the family of BLACKBOX (Kautz and Selman 1999) classical planners. They simultaneously code a planning graph (Blum and Furst 1995) and temporal constraints, then call the solver (LP, CSP, SMT or SAT) to find a solution.

Spatial Planners

Spatial knowledge is essential in planning. For instance, in the case of building evacuation, to achieve this mission the agents have to move through the environment between the different areas. In literature, there are somme planners exploiting spatial knowledge. For example: ASYMOV (Gravot, Cambon, and Alami 2005), one presented in (Guitton et al. 2008) and SPOON (Belouaer, Bouzid, and Mouaddib 2011).

The planner ASYMOV computes plans for handling problems in which the execution of an action has an important effect on the spatial representation of the problem. For example, when an agent has to carry an object, the shape of the whole agent with the object is different from that of the agent load. The planner described in (Guitton et al. 2008) consists of two modules reasoning: a symbolic reasoning module supported by a planner task and a reasoning module supported by a path planner.

These two planners manage only the quantitative spatial information (numeric distance, coordinates...). This is not enough. For instance, to move through an environment requires a spatial knowledge for describing it. In this kind of application, it is also necessary to know the adjacency relations, the distances between the different regions to compute a path. Spatial knowledge can be quantitative (numerical), qualitative (topological or fuzzy) and hierarchical (simplifies the description of the global environment). Also, These two planners do not express planning problems taking into account the spatial dimension.

SPOON is an hybrid planner combining two reasoning modules. Symbolic reasoning module supported by task planner. Spatial reasoning module supported by path planner and *SpaceOntology*. This ontology provides a structured knowledge of the explored environment in the planning process. SPOON uses Space-PDDL as a language to express a planning problem taking into account the spatial dimension.

Our main contribution is to integrate some concepts of *SpaceOntology* in TLP-GP-2 in order to consider both spatial and temporal dimensions in planning problem.

Preliminaries

This section presents all required notations for the definition of spatio-temporal rules.

A $fluent$ is a positive or negative atomic proposition. We define a set of spatial predicates leading to spatial fluents when instantiated. We consider conditions on the value of fluents and changes of this value may either instantaneous or be imposed over an interval. An action a is a quadruple $< Cond(a), Eff(a), Constr(a), Mov(a) >$, where:

- $Cond(a)$ is a set of fluents which are required to be true for a to be executed,
- $Eff(a)$ is the set of fluents which are established or destroyed by the execution of a,
- $Constr(a)$ is a set of constraints between the relative times of events which occur during the execution of a,
- $Mov(a)$ is a set of vectors of real valued functions associated with each of spatial fluent $Move(e)$ in $Eff(a)$ corresponding with the movement of spatial entity e produced by a.

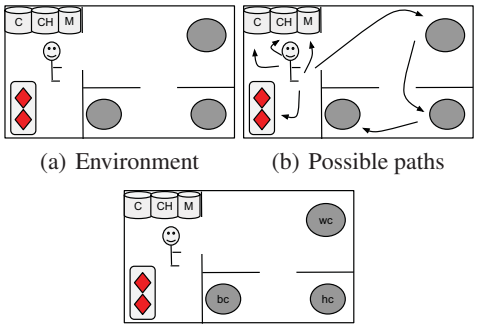
An event corresponds to one of nine possibilities: the instant when, or the beginning or end of an interval over which a fluent is required or produced or destroyed by an action a. We use respectively the notation $a \rightarrow f$ to denote the event that action a establishes fluent f, $a \rightarrow \neg f$ to denote the event

that action a destroys fluent f, and $f \rightarrow a$ to denote the event that a requires the fluent f. When these events occur over an interval, we use respectively $a \mid\rightarrow f$, $a \mid\rightarrow \neg f$ and $f \mid\rightarrow a$ to denote the beginning of this interval, and respectively $a \rightarrow\mid f$, $a \rightarrow\mid \neg f$, $f \rightarrow\mid a$ to denote its end. We use the notation $\tau(E)$ to represent the time in a plan at which an event E occurs. For a given action a, let $Events(a)$ represents a different events which constitute its definition. If A is a set of actions, then $Events(A)$ is the union of the $Events(a)$ (for all $a \in A$).

For any spatial entity e we define $T\delta(e)$ as a set of temporal variables corresponding to a spatial event on e in the actions definition. These temporal variables correspond to a reading of the spatial state of e for the conditions (numeric position, numeric distance, fuzzy distance) or an update of this state for effects (movement, hierarchical link). The set of all temporal variables corresponding to the spatial events on all of the entities of SpE is denoted $T\delta$.

Case Study

To illustrate our claim let us consider the following example. A waiter is involved in the environment shown in Figure 1(a). His mission is to serve each customer with the ordered drinks : a white coffee, a black coffee and a hot chocolate (Figure 1(c)).



(a) Environment (b) Possible paths

(c) Waiter's goal

Figure 1: Environment and mission.

To achieve this mission the waiter considers the spatial description of the environment in which he operates in order to find a plan by which he prepares and serves drinks (Figure 1(b)). Also, the success of this mission requires the satisfaction of the following temporal constraint: when drink is served it is still warm enough. Let us consider that the waiter's goal is to serve the white coffee. In the following, we represent the description of its mission in PDDL extended with spatial fluents (Figure 2, Figure 3). Table 1, Table 2,Table 4 and Table 5 define some elements which are useful for understanding this paper and defined in our study case.

Table 1 defines types of used objects in our problem. Table 2 presents used objects. Table 4 presents the different symbolic or spatial fluents necessary. Table 5 presents the various actions that the waiter can execute.

```
(define (domain service)
 (:requirements …)
 (:types …)
 (:predicates …)
 (:durative-action Pick
    :parameters (?ingredient - ingredient ?area - area)
    :duration (= ?duration (pick-time ?ingredient))
    :condition (and (at start (on ?ingredient ?area))
                    (over [start end[ (at ?area))
                    (at start (FD_Near waiter ?area)))
    :effect (and (at end (carry ?ingredient)))

 (:durative-action Go
     :parameters (shelf table)
     :duration (= ?duration (travel-time shelf table))
     :condition (and (at start (at shelf))
                     (at start (Inside waiter kitchen))
     :effect (and (at end (at table)
              (over [start (+ start (/ (?duration) 3))[
             (Move_1 waiter : Delta_x[t]=2.0*t : Delta_y[t]=0.0*t))
             (at (+ start (/ (?duration) 3)) (not (Inside kitchen)))
             (at (+ start (/ (?duration) 3)) (Inside lounge))
             (over [(+ start (/ (?duration) 3)) end[
             (Move_2 waiter : Delta_x[t]=1.5*t : Delta_y[t]=1.0*t))
 ))

  (:durative-action  Heat …)
  (:durative-action Make_W_C …)
  (:durative-action Serve …))
```

Figure 2: Domain definition in PDDL.

```
(:init
      (Inside waiter kitchen)
      (Inside table lounge)
      (= table.center.x 4.0)
      (= table.center.y 3.2)
      …
      (at shelf)
      (on coffee shelf)
      (on milk shelf)
      (= (pick-time coffee) 0.2)
      (= (pick-time milk) 0.2)
      (= (travel-time shelf table) 3.0)
...)
(:goal
      (at 20 (served table cup))
      (at 20 (contains2 cup coffee milk))
      (over [20 25] (hot coffee))
      (over [20 25] (hot milk))
)
```

Figure 3: Mission definition in PDDL.

Definition 1 (Spatio-temporal planning problem) *A spatio-temporal planning problem $< I, A, G >$ consists of a set of actions A, an initial state I and a goal state G, where I and G are sets of fluents.*

We introduce three basic constraints that all spatio-temporal plans must satisfy :

- *Inherent constraints* on the set of actions A: for all $a \in A$, a satisfies $Constr(a)$.
- *Contradictory-effects constraints* on the set of actions A : for all $a_i, a_j \in A$, for all fluents f such that $\neg f \in Eff(a_i)$ and $f \in Eff(a_j)$, $\tau(a_i \rightarrow\mid \neg f) < \tau(a_j \mid\rightarrow f) \vee \tau(aj \mid\rightarrow f) < \tau(ai \rightarrow\mid \neg f)$.
- *Contradictory-movements constraints* on the set of actions A: for all $a_i, a_j \in A$, for all $e \in SpE$, for all spatial fluents $Move(e)$ such that $Move(e) \in Eff(a_i)$ and $Move(e) \in Eff(a_j)$, $\tau(a_i \rightarrow\mid Move(e)) < \tau(a_j \mid\rightarrow Move(e)) \vee \tau(a_j \mid\rightarrow Move(e)) < \tau(a_i \rightarrow\mid Move(e))$.

Types	Semantic
$spatial\_entity$	any spatial object
$object$	any symbolic object
$area\text{- }spatial\_entity$	an area in the environment
$ingredient\text{ - }object$	an ingredient used to make a drink
$drink\text{ - }object$	ordered or ready drink
$room\text{ -}spatial\_entity$	an office that may include areas

Table 1: Types

Objects	Semantic
$shelf\text{- }area$	a storage area
$table\text{-}area$	area in which is the table
$cooker\text{- }area$	a cooking area
$kitchen\text{ - }room$	area that includes $shelf$ and $cooker$
$lounge\text{ - }room$	area that includes $shelf$ and $cooker$
$milk\text{ - }ingredient$	milk ingredient
$coffee\text{ - }ingredient$	coffee ingredient
$cup\text{ - }spatial\_entity$	container for receiving a drink
$waiter\text{ - }spatial\_entity$	agent that serves the tables

Table 2: Objects

Fluents	Semantic
$at(area)$	waiter's position
$on(shelf, ingredient)$	the ingredient is on the shelf
$carry(object)$	the waiter carries a drink
$hot(object)$	the ingredient or the drink is hot

Table 3: Symbolic fluents

Spatial fluents	Semantic
$Inside$ $(spatial\_entity_1, spatial\_entity_2)$	$spatial\_entity_1$ is included in $spatial\_entity_2$
$Move(spatial\_entity)$	indicates the existence of a displacement
FD_{label} $(spatial\_entity_1, spatial\_entity_2)$	indicates the fuzzy distance between two spatial entities

Table 4: Spatial fluents

Actions	Semantic
$Go(spatial\_entity_1, spatial\_entity_2)$	move from $spatial\_entity_1$ to $spatial\_entity_2$
$Pick(ingredient, area)$	take an ingredient from an area
$Heat(ingredient, area)$	heat an ingredient in an area
$Serve(area, cup)$	serve a cup to the customer
$Make\_W\_C(ingredient, ingredient)$	prepare the white coffee

Table 5: Actions

Definition 2 (Spatio-temporal plan) *$P = < A, t >$ where A is a set of action-instances $\{a_1, \ldots, a_n\}$ and t is a real-valued function on $Events(A)$, is a spatio-temporal plan for the problem $< I, A', G >$ if (1) $A \subset A'$ and (2) P satisfies the inherent and contradictory-effect constraints on A; when p is executed (i.e. fluents are established or destroyed at the times given by t) starting from the initial state I and (3) for all $a_i \in A$, each $f \in Cond(a_i)$ is true when it is required and (4) all goal fluents $g \in G$ are true at the end of execution of P.*

To encode a planning problem we use a simplified planning graph *GP* (Blum and Furst 1995). *NodeA* denotes the set of actions nodes of *GP*. *ArcsCond* and $ArcsEff$ respectively denote the set of conditions arcs and the effects arcs of *GP*. *SpE* denotes the set of spatial entities defined in the planning problem. In the following section we present the set of rules defining the ST-SMTPLAN.

ST-SMTPLAN: Encoding Rules for Spatio-Temporal Planning

To encode a spatio-temporal planning problem, we define three types of rules: (1) rules (R) encode the plan solution structure of the problem based on a temporally extended planning graph, (2) rules (T) manage temporal knowledge and (3) rules (S) manage spatial and temporal knowledge.

Formally, $\mathcal{R}$ denotes the set of ST-SMTPLAN rules $\mathcal{R}$ ($\mathcal{R} = R \cup T \cup S$). In this paper, we present a subset of $\mathcal{R}$ which are used to solve the problem defined in the study case. The rules presented here are classified by group. Those concerning the definition: initial and goal states, the graph temporally extended, the spatial and temporal dimensions simultaneously and temporally extended mutexes.

Initial State and Goal

We define two dummy actions *Init* and *Goal*. *Init* produces the initial state and *Goal* produces the goal state. This two dummy actions are both true.

$$Init \wedge Goal \qquad \text{(R1)}$$

This part presents some of S1 rules for encoding the description of the initial state and goal. From a spatial perspective, the initial state considers the description of positions (rule S1.1) and hierarchical definition (rule S1.2). Intuitively, the instant at which the goal state is true is later than the instant at which the initial state is true. The rule (T1) encodes this intuition.

(S1.1): Numeric position of entities in the initial state In *SpaceOntology*: a spatial entity e is a localized entity in a given space. It can be static; any element fixed in space which position changes only by an action performed by an agent. Or it can be dynamic; any entity which spatial position changes over time in space. A spatial entity is defined by its center.

$$\bigwedge_{e \in SpE} \begin{pmatrix} (x[\tau(Init)](e) = e.center.x) \\ (y[\tau(Init)](e) = e.center.y) \end{pmatrix} \quad \text{(S1.1)}$$

Example 1 *Let us consider the space defined in Figure 1(c). We focus on the table which is in the lounge.*

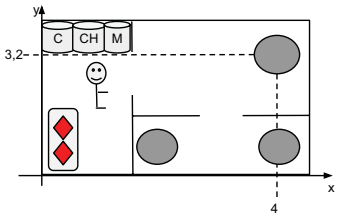


Figure 4: Spatial entites positions.

To locate this table, we consider its center and project it on both axis (Figure 4). This allows us to instantiate the rule (S1.1) as follows:

$$\begin{cases} x[\tau(Init)](table) = 4 \\ y[\tau(Init)](table) = 3,2 \end{cases}$$

Remark 1 *We assume that we are in 2D space. x denotes x-axis and y denotes y-axis. All rules are defined for x-axis and y-axis. In this paper, we detail only rules for x-axis. Rules can be extend to 3D.*

(S1.2): Inclusion relation of entities in the initial state *SpaceOntology* permits the hierarchical representation in order to simplify the description of the global environment. To represent the hierarchical relation, we define the spatial fluent *Inside*. This spatial fluent expresses the inclusion link between two spatial entities. Formally, for each pair of spatial entities (e_1, e_2) where the relation $e_1 \subset e_2$ is true, then we add the following rule:

$$\bigwedge_{(e_1,e_2) \in SpE^2 / e_1 \subset e_2} Inside[\tau(Init)](e_1, e_2) \quad \text{(S1.2)}$$

Example 2 *The initial state of our problem is depicted in Figure 1(a).* $\tau(Init)$ *is the instant of the initial time.* $Inside[\tau(Init)](waiter, kitchen)$ *expresses that the waiter is in the kitchen in the initial state. Thus, to express spatial inclusion in the initial state we instantiate the rule S1.2 as follow:*

$$\begin{cases} Inside[\tau(Init)](shelf, kitchen) \\ Inside[\tau(Init)](cooker, kitchen) \\ Inside[\tau(Init)](waiter, kitchen) \end{cases}$$

(T1): Lower and upper bounds The initial instant (when propositions of the initial state are true) precedes every instant of the beginning of the preconditions of the other actions. The final instant (when the goal propositions are true) follows all instants of the end of the effects of the other actions.

$$(\tau(Init) \leq \tau(Goal)) \wedge \bigwedge_{a \in NodeA} \begin{pmatrix} (\tau(Init) \leq \underset{p \in Cond(a)}{Min} \{\tau(p \mid\rightarrow a)\}) \\ \wedge (\underset{q \in Eff(a)}{Max} \{\tau(a \rightarrow\mid q)\} \leq \tau(Goal)) \end{pmatrix} \quad \text{(T1)}$$

Encoding of Temporally Extended Planning Graph

The rules (R) encode the problem as temporally extended planning graph. They encode the causal links between conditions (R2) and effects and actions selections (R3).

(R2): Conditions production by causal links If an action b is active in the plan, then for each of its preconditions p, it exists at least one causal link (noted $Link(a, p, b)$) from the action a (which produces this precondition) to the action b.

$$\bigwedge_{(p,b) \in ArcsCond} \begin{pmatrix} b \Rightarrow \bigvee_{(a,p) \in ArcsEff} Link(a, p, b) \end{pmatrix} \quad \text{(R2)}$$

(R3): Actions activation and partial order If a causal link exists between an action a which produces a precondition p for an action b, then a and b are actives in the plan. Moreover, the instant when a certainly produces p precedes or is the same than the instant when b begins to need p.

$$\bigwedge_{(a,p) \in ArcsEff} \bigwedge_{(p,b) \in ArcsCond} \begin{pmatrix} Link(a, p, b) \\ \Rightarrow \begin{pmatrix} a \wedge b \\ \wedge (\tau(a \mid\rightarrow p) \leq \tau(p \mid\rightarrow b)) \end{pmatrix} \end{pmatrix} \quad \text{(R3)}$$

Encoding Spatial Knowledge

An action may require the satisfaction of spatial relations between two spatial entities for execution. An action can change the spatial definitions between two spatial entities. Spatial information is integrated into the sets $Cond(a)$ and $Eff(a)$ when the action a is defined in the planning domain.

A spatial entity may be dynamic or static. The spatial entity e is dynamic, the use of the predicate $Move(e)$ in the effects of the action a corresponds to the movement of the spatial entity e by the action a on a given temporal interval $[t_1, t_2]$. With the fluent $Move(e)$ we associate a moving function. We define on the interval $[t_1, t_2]$ a linear function corresponding to the movement of spatial entity e for each axis. For horizontal axis (ox) the linear function is denoted by $\Delta_x^a[t](e) = v_x^a(e) \times t$. The global movement of the entity e produced by an action a on the x-axis is denoted by $\Delta_x^a(e)$ ($\Delta_x^a(e) = \Delta_x^a[\tau(a \rightarrow\mid Move(e)) - \tau(a \mid\rightarrow Move(e))](e)$) .

Spatial Conditions The rules S2 encode the spatial conditions. Indeed, it is necessary to compute distances (S2.1) and to know numeric distances or approximative distances (S2.2) in order to execute actions or not.

(S2.1): Computing numeric distances between entities For each couple of spatial entities (e_1, e_2) we denote by $d_x[t]\{e_1, e_2\}$ a numeric distance between e_1 and e_2 according to x-axis.

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{t\in T\delta(e_1)\cap T\delta(e_2)} \begin{pmatrix} (x[t](e_1) \leq x[t](e_2) \Rightarrow d_x[t]\{e_1,e_2\} = x[t](e_2) - x[t](e_1) \\ (x[t](e_2) < x[t](e_1) \Rightarrow d_x[t]\{e_1,e_2\} = x[t](e_1) - x[t](e_2) \end{pmatrix} \quad \text{(S2.1)}$$

(S2.2): Numeric distance between entities The spatial fluent $Distance(e_1, e_2) = value$ where value is a rational number can be used as a constraint in the definition of $Cond(a)$. It represents a condition of fixed numeric distance between two spatial entities e_1 and e_2. A constraint $d_x[t]\{e_1, e_2\} = value$ is then added for each action a of the graph at $t = \tau(Distance(e_1, e_2) = value \rightarrow a)$. We can generalize by replacing equality by inequality.

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{a\in NodeA/(Distance(e_1,e_2)=value)\in Cond(a)} a \Rightarrow \left(d_x[\tau(Distance(e_1,e_2) = value \rightarrow a]\{e_1,e_2\} = value\right) \quad \text{(S2.2)}$$

(S2.3): Fuzzy distance between entities *SpaceOntology* defines four concepts $\{Near, NearEnough, FarEnough, Far\}$ in order to express fuzzy distance. The special fluent $FD_{label}(e_1, e_2)$ represents a fuzzy distance between two entities e_1 and e_2 such as $label = \{Near, NearEnough, Far, FarEnough\}$. Each label is associate to an interval $[\alpha_{label}, \beta_{label}[$.

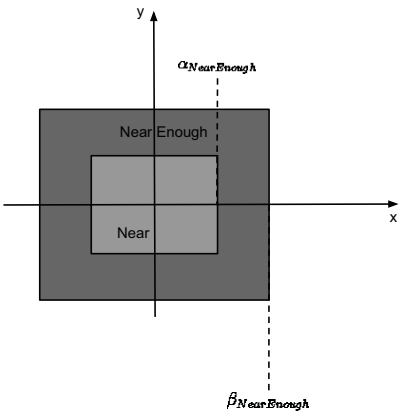


Figure 5: Fuzzy distances zones relatives to a spatial entity.

For example, let us consider the fuzzy distance near enough, so $label = NearEnough$ and the associated interval is $[\alpha_{NearEnough}, \beta_{NearEnough}[$ (Figure 5).

$$\bigwedge_{(e_1,e_2)\in SpE^2/(e_1\neq e_2)} \bigwedge_{a\in NodeA/FD_{label}(e_1,e_2)\in Cond(a)} a \Rightarrow \left(\alpha_{label} \leq d_x[\tau(FD_{label}(e_1,e_2) \rightarrow a)]\{e_1,e_2\} < \beta_{label}\right) \quad \text{(S2.3)}$$

Example 3 *Let us consider our study case. We define the following variables:*

$$\begin{cases} a = Pick(milk, shelf) \\ t = \tau(FD_{Near}(waiter, shelf) \rightarrow Pick(milk, shelf)) \\ e_1 = waiter \\ e_2 = shelf \end{cases}$$

The application of the rule (S2.1) computes the distance between the waiter and the shelf:

$$\begin{cases} (x[t](waiter) \leq x[t](shelf) \Rightarrow \\ \quad d_x[t]\{waiter, shelf\} = x[t](shelf) - x[t](waiter) \\ (x[t](shelf) < x[t](waiter) \Rightarrow \\ \quad d_x[t]\{waiter, shelf\} = x[t](waiter) - x[t](shelf) \end{cases}$$

Let us consider that the execution of the action Pick requires that the distance between the waiter and the shelf is 0.3. The application of the rule (S2.2) checks the distance between the waiter and the shelf:

$$\begin{cases} Pick(milk, shelf) \\ \quad \Rightarrow \\ (d_x[\tau(Distance(waiter, shelf) = 0.3 \rightarrow Pick(milk, shelf))]\{waiter, shelf\} \\ \quad = 0.3) \end{cases}$$

The application of the rule (S2.3) checks the fuzzy distance between the waiter and the shelf:

$$\begin{cases} Pick(milk, shelf) \\ \quad \Rightarrow \\ \alpha_{Near} \leq \\ d_x[\tau(FD_{Near}(waiter, shelf) \rightarrow Pick(milk, shelf))]\{waiter, shelf\} < \\ \quad \beta_{Near} \end{cases}$$

Spatial effects Computing distances between entities requires to know their numerical position. It is therefore necessary to know these positions in the initial state, but also to recalculate when the effect of an action changes.

(S3): Computing new positions Let us consider the temporal interval $[t_1, t_2]$. The spatial entity e_1 position at time t_2 is calculated from its position at time t_1 and from the movements $\delta_x^{Ind}[t_1, t_2](e_1, a, e_2)$ induced by all entities e_2 that can be moved by an action a.

$$\bigwedge_{e_1 \in SpE} \bigwedge_{(t_1,t_2) \in T\delta^2} \left(\begin{array}{c} (x[t_2](e_1) = x[t_1](e_1) + \\ \sum_{e_2,a \in SpE \times NodeA/\Delta_x^a(e_2) \neq 0} \delta_x^{Ind}[t_1,t_2](e_1,a,e_2) \end{array} \right) \quad \text{(S3)}$$

Example 4 *Let us consider the following action $Go(shelf, table)$. Note by :*

- $t_1 = \tau(Go(shelf, table) \mid\!\rightarrow Move_1(waiter))$
- $t_2 = \tau(Go(shelf, table) \rightarrow\!\mid Move_2(waiter))$

Let us apply rule S3.

$$x[t_2](cup) = \left\{ \begin{array}{l} x[t_1](cup) \quad + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(shelf, cooker), waiter) + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(cooker, shelf), waiter) + \\ \delta_x^{Ind}[t_1,t_2](cup, Go(shelf, table), waiter) \quad + \\ \ldots \end{array} \right.$$

To compute the induced movement of an entity e on other entities or on entity e, it is necessary to know the hierarchical relation between e and all other entities at the instant when e starts or ends its movement. We can now describe the rules considering hierarchical links.

(S4.1): Production and removal of hierarchical link If an action a which produces the inclusion of a spatial entity e_1 in a spatial entity e_2 at time $t = \tau(a \rightarrow\!\mid Inside(e_1, e_2))$ is active in the plan then the predicate $Inside[t](e_1, e_2)$ corresponding to the inclusion $e_1 \subset e_2$ at time t is true.

$$\bigwedge_{(e_1,e_2) \in SpE^2/e_1 \neq e_2} \bigwedge_{a \in NodeA/Inside(e_1,e_2) \in Eff(a)} \big(a \Rightarrow Inside[\tau(a \rightarrow\!\mid Inside(e_1, e_2))](e_1, e_2)\big) \quad \text{(S4.1.1)}$$

If an action a which removes the inclusion $e_1 \subset e_2$ at time $t = \tau(a \mid\!\rightarrow \neg Inside(e_1, e_2))$ is active in the plan then the predicate $Inside[t](e_1, e_2)$ is false.

$$\bigwedge_{(e_1,e_2) \in SpE^2/e_1 \neq e_2} \bigwedge_{a \in NodeA/\neg Inside(e_1,e_2) \in Eff(a)} \big(a \Rightarrow \neg Inside[\tau(a \mid\!\rightarrow \neg Inside(e_1, e_2))](e_1, e_2)\big) \quad \text{(S4.1.2)}$$

Example 5 *The waiter's movement between the shelf and the table ($Go(shelf, table)$) has two spatial effects on hierarchy. The first is the production of the hierarchical link "the waiter is inside the lounge" (applying the rule (S4.1.1)):*

$$\left\{ \begin{array}{l} Go(shelf, table) \Rightarrow \\ \quad Inside[\tau(Go(shelf, table) \rightarrow\!\mid Inside(waiter, lounge))](waiter, lounge) \end{array} \right.$$

The second is the destruction of the hierarchical link "the waiter is inside the kitchen" (applying the rule (S4.1.2)):

$$\left\{ \begin{array}{l} Go(shelf, table) \Rightarrow \\ \quad \neg Inside[\tau(Go(shelf, table) \mid\!\rightarrow \neg Inside(waiter, kitchen))](waiter, kitchen) \end{array} \right.$$

(S4.2): Propagation of hierarchical links in time If a hierarchical link $e_1 \subset e_2$ is active in the plan at time $t \in T\delta$, when a spatial event occurs, then there exists at least one positive hierarchical link protection interval (denoted $LinkInside(e_1, e_2, a, t)$) of an action a, which produces $Inside(e_1, e_2)$, up to time t.

$$\bigwedge_{(e_1,e_2) \in SpE^2/e_1 \neq e_2} \bigwedge_{t \in T\delta} \left(\begin{array}{c} Inside[t](e_1, e_2) \\ \Rightarrow \bigvee_{a \in NodeA/Inside(e_1,e_2) \in Eff(a)} LinkInside(e_1, e_2, a, t) \end{array} \right) \quad \text{(S4.2.1)}$$

$$\bigwedge_{(e_1,e_2) \in SpE^2/e_1 \neq e_2} \bigwedge_{a \in NodeA/Inside(e_1,e_2) \in Eff(a)} \bigwedge_{t \in T\delta} \left(\begin{array}{c} LinkInside[t](e_1, e_2, a, t) \\ \Rightarrow a \wedge Inside[t](e_1, e_2) \wedge (\tau(a \rightarrow\!\mid Inside(e_1, e_2)) \leq t) \end{array} \right) \quad \text{(S4.2.2)}$$

Similarly, if a hierarchical link $e_1 \subset e_2$ is not active in the plan at time $t \in T\delta$, when a spatial event occurs, then there exists at least one negative hierarchical link protection interval (denoted $LinkNotInside(e_1, e_2, a, t)$) of an action a, which produces $\neg Inside(e_1, e_2)$, up to time t.

Example 6 *We assume that t is the instant when the action serve ($Serve(table, cup)$) requires that the waiter has the cup ($t = \tau(Inside(cup, waiter) \mid\!\rightarrow Serve(table, cup))$). If at the instant t, the waiter has the cup then it exists a link to propagate this property from the action $Make\_W\_C(milk, coffee, cup)$ which produces it.*

$$\left\{ \begin{array}{l} Inside[t](cup, waiter) \\ \quad \Rightarrow LinkInside(cup, waiter, Make\_W\_C(milk, coffee, cup), t) \end{array} \right.$$

If a link propagates the fact that the waiter has the cup from the action $Make\_W\_C(milk, coffee, cup)$, up to the instant t, then this action must be active in the plan and produces the hierarchical link before the instant t.

$$\left\{ \begin{array}{l} LinkInside(cup, waiter, Make\_W\_C(milk, coffee, cup), t) \\ \quad \Rightarrow Make\_W\_C(milk, coffee, cup) \wedge Inside[t](cup, waiter) \\ \wedge \tau(Make\_W\_C(milk, coffee, cup) \rightarrow\!\mid Inside[(cup, waiter) \leq t \end{array} \right.$$

We consider some assumptions. We define them as rules. In the following, we present some of them. Rule (S5.1.1) requires that an entity can't enter in another entity when this later is moving. Similarly, rule (S5.1.2) states that an entity exit from another entity when this later is moving. Rule (S5.2) states that each entity is included in itself.

$$\bigwedge_{(e_1,e_2) \in SpE^2/e_1 \neq e_2} \bigwedge_{(a,b) \in NodeA^2/Inside(e_1,e_2) \in Eff(a) \wedge \Delta_x^b(e_2) \neq 0} \left(\begin{array}{c} a \wedge b \Rightarrow \\ (\tau(a \rightarrow\!\mid Inside(e_1, e_2)) < \tau(b \mid\!\rightarrow Move(e_2)) \\ \vee (\tau(b \rightarrow\!\mid Move(e_2) < \tau(a \mid\!\rightarrow Inside(e_1, e_2)) \end{array} \right) \quad \text{(S5.1.1)}$$

$$\bigwedge_{e \in SpE} \bigwedge_{t \in T\delta} \big(Inside[t](e, e)\big) \quad \text{(S5.2)}$$

Example 7 *Let us consider the two actions: prepare the coffee ($Make\_W\_C(milk, coffee, cup)$) and move between the cooker and the shelf ($Go(cooker, shelf)$). In order to execute these two actions, we must respect one of this situation (applying the rule (S5.1.1)) :*

- *the instant when the action $Make\_W\_C(milk, coffee, cup)$ finish to produce that the waiter has the cup is before the instant when the action $Go(cooker, shelf)$ begin to move the waiter.*
- *the instant when the action $Go(cooker, shelf)$ finish to move the waiter between the cooker and the shelf is before the instant when the action $Make\_W\_C(milk, coffee, cup)$ begins to produce that the waiter has the cup.*

$$\left\{\begin{array}{c} Make\_W\_C(milk,coffee,cup) \wedge Go(cooker,shelf) \\ \Rightarrow \\ \left\{\begin{array}{c} \left\{\begin{array}{c} \tau(Make\_W\_C(milk,coffee,cup) \rightarrow\mid Inside(cup,waiter)) \\ < \tau(Go(cooker,shelf) \mid\rightarrow Move(waiter)) \end{array}\right\} \\ \vee \\ \left\{\begin{array}{c} \tau(Go(cooker,shelf) \rightarrow\mid Move(waiter)) \\ < \tau(Make\_W\_C(milk,coffee,cup) \mid\rightarrow Inside(cup,waiter)) \end{array}\right\} \end{array}\right. \end{array}\right.$$

We can now calculate the induced movements by one entity over another.

(S6.1): Induced movement with hierarchical link We assume an action that a which produces a movement of a spatial entity e_2 ($\Delta_x^a(e_2) \neq 0$)is active in the plan. We consider a spatial entity e_1 such as $e_1 \subset e_2$. Also, we consider the temporal interval $I = [t_1, t_2]$. Let $[t, t']$ the intersection of I and the interval on which e_2 movement is produced by a. If this intersection is not empty then the movement of e_1 induced by e_2 is $\delta_x^{Ind}[t_1, t_2](e_1, a, e_2)$ is equal to $\Delta_x^a[t' - t](e_2)$, else the induced movement is zero.

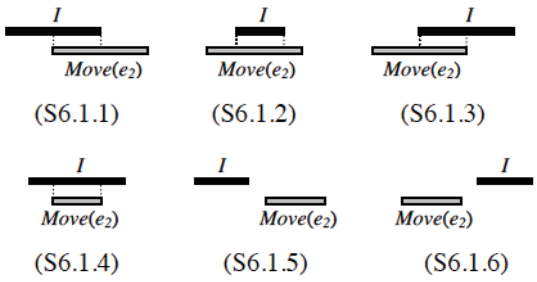


Figure 6: Possible configurations between an interval and spatial entity's movement.

The different configurations and the associated rules are illustrated by Figure 6. Here we detail only the rule (S6.1.1).

$$\bigwedge_{(e_1,e_2)\in SpE^2} \bigwedge_{(t_1,t_2)\in T\delta^2} \bigwedge_{a\in NodeA/\Delta_x^a(e_2)\neq 0} \left(\begin{array}{c} (a \wedge Inside[\tau(a \mid\rightarrow Move(e_2))](e_1,e_2) \\ \wedge t_1 \leq \tau(a \mid\rightarrow Move(e_2)) \wedge \tau(a \mid\rightarrow Move(e_2)) \leq t_2 \\ \wedge t_2 \leq \tau(a \rightarrow\mid Move(e_2)) \\ \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2) = \Delta_x^a[t_2 - \tau(a \mid\rightarrow Move(e_2))](e_2)) \end{array}\right) \quad \text{(S6.1.1)}$$

Example 8 *Let us denote by:*

- $t_1 = \tau(Go(shelf, table) \mid\rightarrow Move_1(waiter))$
- $t_2 = \tau(Go(shelf, table) \rightarrow\mid Move_2(waiter))$

We are in the case of rule (S6.1.4). On the global movement interval $[t_1, t_2]$ over which the waiter moves from the shelf to the table. If the waiter has the cup, each of the movement ($Move_1(waiter)$ and $Move_2(waiter)$) induces the movement on the cup.

$$\left\{\begin{array}{l} Go(shelf,table) \wedge \\ Inside[\tau(Go(shelf,table) \mid\rightarrow Move(waiter))](cup,waiter) \wedge \\ t_1 \leq \tau(Go(shelf,table) \mid\rightarrow Move(waiter)) \wedge \\ \tau(Go(shelf,table) \rightarrow\mid Move(waiter)) \leq t_2 \wedge \\ \Rightarrow (\delta_x^{Ind}[t_1,t_2](cup, Go(shelf,table), waiter) = \\ \quad \Delta_x^{Go(shelf,table)}[t_2 - \tau(Go(shelf,table) \mid\rightarrow Move(waiter))](waiter)) \end{array}\right.$$

(S6.2): Induced movement without hierarchical links If an action a which moves an entity e_2 with displacement $\Delta_x^a(e_2)$ is active in the plan and we have a spatial entity e_1 distinct from e_2 ($e_1 \neq e_2$) and e_1 is not inside e_2 ($e_1 \not\subset e_2$) when a starts to move e_2 then the displacement of e_1 on the interval $[t_1, t_2]$ induced by the movement of e_2 produced by the action a: $\delta_x^{Ind}[t_1, t_2](e_1, a, e_2)$ is zero.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2} \bigwedge_{(t_1,t_2)\in T\delta^2} \bigwedge_{a\in NodeA/\Delta_x^a(e_2)\neq 0} \left(\begin{array}{c} (a \wedge \neg Inside[\tau(a \mid\rightarrow Move(e_2))](e_1,e_2) \\ \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2) = 0) \end{array}\right) \quad \text{(S6.2)}$$

Example 9 *In our study case; the waiter moves from shelf to table without taking the cup ($cup \not\subset waiter$). The application of (S6.2) shows that the displacement of the cup induced by waiter's movement is zero.*

$$\left\{\begin{array}{l} Go(shelf,table) \wedge \\ \neg Inside[\tau(Go(shelf,table) \mid\rightarrow Move(waiter))](cup,waiter) \\ \qquad \Rightarrow (\delta_x^{Ind}[t_1,t_2](cup, Go(shelf,table), waiter) = 0 \end{array}\right.$$

(S6.3): Induced movement by an inactive action If an action a which produces a movement of a spatial entity e_2, noted $\Delta_x^a(e_2)$, is not active in the plan and given a spatial entity e_1 (not necessarily distinct from e_2) then the movement of e_2, noted $\delta_x^{Ind}[t_1, t_2](e_1, a, e_2)$, on the temporal interval $[t_1, t_2]$ induced by the movement of e_2 produced by a is zero.

$$\bigwedge_{(e_1,e_2)\in SpE^2}\ \bigwedge_{(t_1,t_2)\in T\delta^2 a\in NodeA/\Delta_x^a(e_2)\neq 0} \Big(\neg a \Rightarrow (\delta_x^{Ind}[t_1,t_2](e_1,a,e_2)=0)\Big) \tag{S6.3}$$

Temporally extended mutexes

Two propositions are mutually exclusive when:

- the propositions are antagonists, for instance p and $\neg p$;
- or the propositions represent the movement of the same spatial entity e (the special predicate $Move(e)$ is used by two actions).

In these cases, the propositions can not occur on the same temporal interval.

(R4): Temporally extended mutexes If a causal link protects a proposition p and an action produces its is active in the plan, then the temporal interval corresponding to the causal link and the temporal interval corresponding to activation of $\neg p$ by the action are disjunctive.

$$\bigwedge_{(a,p)\in ArcsEff}\ \bigwedge_{(p,b)\in ArcsCond}\ \bigwedge_{(c,\neg p)\in ArcsEff} \left(\begin{array}{c} Link(a,p,b)\wedge c \\ \Rightarrow (\tau(c\rightarrow|\ \neg p)<\tau(a\ |\rightarrow p))\vee(\tau(p\rightarrow|\ b)<\tau(c\ |\rightarrow\neg p)) \end{array}\right) \tag{R4.1}$$

If two actions respectively producing a proposition p and its negation are active in the plan, then the temporal intervals corresponding to the activation of p and activation of $\neg p$ are disjunctive.

$$\bigwedge_{(a,p)\in ArcsEff}\ \bigwedge_{(b,\neg p)\in ArcsEff} \left(\begin{array}{c} a\wedge b \\ \Rightarrow (\tau(b\rightarrow|\ \neg p)<\tau(a\ |\rightarrow p))\vee(\tau(a\rightarrow|\ p)<\tau(b\ |\rightarrow\neg p)) \end{array}\right) \tag{R4.2}$$

Mutexes of actions that move the same entity If a causal link protects the proposition $Move(e)$ which represents a spatial entity e moving and also an action produces the proposition $Move(e)$ is active in the plan then the temporal interval corresponding to the causal link and the temporal interval corresponding to the activation of $Move(e)$ by the action are disjunctive. If two actions produce $Move(e)$ are active in the plan, then the temporal intervals associated with activation of $Move(e)$ by these actions are disjunctive. To obtain respectively the rules (S7) corresponding to these kind of mutual exclusion, it suffices to replace p and $\neg p$ by $Move(e)$ in rules (R4.1) and (R4.2).

(S8): Mutexes on hierarchical links protection interval Similarly as causal link protection (R4.1), we add two rules, respectively (S8.1) and (S8.2), providing a hierarchical link protection for $LinkInside(e_1,e_2,a,t)$ and $LinkNotInside(e_1,e_2,a,t)$.

$$\bigwedge_{(e_1,e_2)\in SpE^2/e_1\neq e_2}\ \bigwedge_{a\in NodeA/Inside(e_1,e_2)\in Eff(a)}\ \bigwedge_{t\in T\delta}\ \bigwedge_{b\in NodeA/\neg Inside(e_1,e_2)\in Eff(b)} \left(\begin{array}{c} (LinkInside(e_1,e_2,a,t)\wedge b) \\ \Rightarrow (\tau(b\rightarrow|\ \neg Inside(e_1,e_2)<\tau(a\ |\rightarrow Inside(e_1,e_2)) \\ \vee(t<\tau(b\ |\rightarrow\neg Inside(e_1,e_2))) \end{array}\right) \tag{S8.1}$$

Discussion

The encoding rules presented in this paper are sound and complete for all temporally-expressive planning problems expressed in the representation language of TLP-GP extended with spatial fluent definition. This result is given, for the rules encoding the solution-plan structure (R) and the temporal constraints (T), by the fact that TLP-GP is sound and complete, using the transformation method of (Cooper, Maris, and Regnier 2010b) to restore its completeness when handling temporally-cyclic problems. Soundness is preserved when adding the spatio-temporal rules (S). When a solution is found by constraint solver, inherent and contradictory-effects constraints are trivially satisfied by rules defined in TLP-GP. Contradictory-movements constraints are also trivially satisfied by rule (S7.2). Moreover, the propagation, encoded by the rules, of spatial knowledge on the entities (position and hierarchy) at all of the instants when this knowledge is used by actions in plan, guarantees that spatial-fluents are true when required. Hence, by Definition 2, solution gives a valid spatio-temporal plan. We now define spatial-relaxed versions of a spatio-temporal plan as all temporal solution-plans of the temporal planning problem resulting from the spatio-temporal planning problem, after deleting all spatial fluents from initial state, actions definition and goal. If a spatio-temporal plan $< A,t >$ exists for a spatio-temporal planning problem $< I,A\prime,G >$ then all solutions corresponding to spatial-relaxed versions of this plan can be found using only the rules (R) and (T), resulting from completeness of TLP-GP. From all these solutions, the ones corresponding to the initial spatio-temporal planning problem should satisfy all the spatio-temporal constraints at all times when a spatial-fluent is required, that is exactly what is encoded by the whole rules (S). Hence, completeness of the encoding rules follows from the fact that these latter solutions can always be found by the solver.

For simplicity of presentation, we have considered only some restricted aspects of *SpaceOntology*. For example, in the definition of the fuzzy distance, the values of α_{label} and β_{label} are the same for all hierarchical levels. We can easily extend the encoding rules to take into account different values for the fuzzy distance depending on the hierarchical links between entities.

Conclusion

ST-SMTPLAN encoding rules allows us to describe and solve spatio-temporal planning problems. It combines the encoding rules described in TLP-GP-2 and the integration of concepts of *SpaceOntology*. This ontology allows us to obtain an optimal spatial and simplified representation of the

initial state and manipulate fuzzy distances between spatial entities. Moreover, the principle of TLP-GP allows us to obtain a temporal planning system capable of solving problems that require concurrency of actions. This system uses a SMT solver and benefit directly from improvements in this type of solver in terms of performance.

This paper is a theoretical work. In order to prove the efficiency of this encoding, we need to define some spatio-temporal planning benchmarks. To simplify the representation space we considered any spatial entity as a point (this point defines the center). This reduces the expressiveness. In our future work we will focus on the size and shape of each spatial entity. An avenue for future research is to use the principle of TLP-GP-1 which performs the backward search on the planning graph. This will allows us to query *SpaceOntology* and code the extracted spatial knowledge at each action selection. This allows a great spatial expressivity added to the temporal expressivity.

References

Belouaer, L.; Bouzid, M.; and Mouaddib, A. 2010. Ontology based spatial planning for human-robot interaction. In *Temporal Representation and Reasoning (TIME)*, 103–110. IEEE.

Belouaer, L.; Bouzid, M.; and Mouaddib, A. 2011. Spatial knowledge in planning language. *International Conference on Knowledge Engineering and Ontology Development*.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with problems requiring temporal coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 08)*. sn.

Cooper, M.; Maris, F.; and Regnier, P. 2010a. Compilation of a high-level temporal planning language into pddl 2.1. In *Tools with Artificial Intelligence (ICTAI)*, volume 2, 181–188. IEEE.

Cooper, M.; Maris, F.; and Regnier, P. 2010b. Solving temporally-cyclic planning problems. In *Temporal Representation and Reasoning (TIME)*, 113–120. IEEE.

Cushing, W.; Kambhampati, S.; and Mausam, W. 2007. When is temporal planning really temporal. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'07), Hyderabad, India*.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)* 20:61–124.

Gerevini, A.; Saetti, A.; and Serina, I. 2010. Temporal planning with problems requiring concurrency through action graphs and local search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10), to appear*.

Ghallab, M., and Alaoui, A. 1989. Managing efficiently temporal relations through indexed spanning trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, 1297–1303.

Gravot, F.; Cambon, S.; and Alami, R. 2005. asymov: a planner that deals with intricate symbolic and geometric problems. *Robotics Research* 100–110.

Guitton, J.; Farges, J.; Chatila, R.; Arioui, H.; Merzouki, R.; and Abbassi, H. 2008. A planning architecture for mobile robotics. In *AIP Conference Proceedings*, volume 1019, 162.

Hu, Y. 2007. Temporally-expressive planning as constraint satisfaction problems. In *Proceedings of 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 192–199.

Huang, R.; Chen, Y.; and Zhang, W. 2009. An optimal temporally expressive planner: Initial results and application to p2p network optimization. In *Proc. of ICAPS*.

Kautz, H., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *International Joint Conference on Artificial Intelligence*, volume 16, 318–325.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *International Joint Conference on Artificial Intelligence*, volume 14, 1643–1651.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *Proceedings of ICAPS*, volume 3, 51–62.

Maris, F., and Régnier, P. 2008a. Tlp-gp: New results on temporally-expressive planning benchmarks. In *Tools with Artificial Intelligence (ICTAI)*, volume 1, 507–514. IEEE.

Maris, F., and Régnier, P. 2008b. Tlp-gp: Solving temporally-expressive planning problems. In *Temporal Representation and Reasoning*, 137–144. IEEE.

Muscettola, N. 1993. Hsts: Integrating planning and scheduling.

Shin, J., and Davis, E. 2004. Continuous time in a sat-based planner. In *Proceedings of the National Conference on Artificial Intelligence*, 531–536. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Younes, H., and Simmons, R. 2003. Vhpop: Versatile heuristic partial order planner. *J. Artif. Intell. Res. (JAIR)* 20:405–430.

Constraint-Based Allocation of Cloud Resources to Maximize Mission Effectiveness

Mark Boddy*
Adventium Labs
111 Third Avenue South, Suite 100
Minneapolis, MN 55401 USA
mark.boddy@adventiumlabs.com

Abstract

We are concerned with the problem of optimizing network resource allocations to mission tasks. The model includes unreliable network assets, multiple mission tasks and phases, and the possibility of over-provisioning one or more tasks as a means of increasing the likelihood of task success. In this paper, we describe an implemented approach to optimizing network resources so as to optimize the expected utility of the mission. This differs significantly from previous work on cloud and network management, where the objective was to optimize some operational measure of the network itself, rather than the effect of network failures on a specific task. The work described here is preliminary: we describe the problem and the approach, define an architecture, and present the current state of the implementation.

Introduction

We are concerned with the problem of optimizing network resource allocations to mission tasks. The model includes unreliable network assets, multiple mission tasks and phases, and the possibility of over-provisioning one or more tasks as a means of increasing the likelihood of task success. In this paper, we describe an implemented approach to optimizing network resources so as to optimize the expected utility of the mission. This differs significantly from previous work on cloud and network management, where the objective was to optimize some operational measure of the network itself, rather than the effect of network failures on a specific task. The work described here is preliminary: we describe the problem and the approach, define an architecture, and present the current state of the implementation.

Optimizing cloud resource allocations to mission tasks with unreliable network assets and over-provisioning requires active management of cloud resources, using an explicit model of the effect of resource failures on the success of mission tasks, and of the dependence of the overall mission on those tasks. The Allocation of Missions Built on Resource Optimization (AMBORO) system[1] provides a flexible, adaptable, and effective means of optimizing the *expected utility* of cloud resource allocations to mission tasks, given estimates of the reliability of those resources. AMBORO provides a stable, extensible, scalable platform for modeling and solving cloud management problems involving uncertain information.

Using a uniform, constraint-based representation for modeling mission tasks, cloud resources and the current set of assignments of resources to tasks provides several benefits, including easy extension of the model to reflect new types of missions, tasks, network components, and operational requirements such as Quality of Service (QoS) guarantees, as well as providing access to a very wide variety of implemented solvers. Other advantages to a constraint-based approach include the ability to define partial solutions, culprit identification in the case of infeasibilities or other conflicts, incremental solutions as additional tasks are added, and the use of local search methods.

In this paper, we describe AMBORO, including its architecture, previous work on which it builds, the form of the input models, and the optimization problem formulation and solution process. We then conclude by reviewing the current state of the system and some planned, near-term future work.

Related Work

In some sense, there is little or no current work in the area being addressed here, because the problem as defined involves modeling and optimization of brand new capabilities. For example, trading off the costs versus the benefits of various forms of Moving Target Defense (MTD) would requires the existence of models (and the experience to populate those models), describing their costs and effectiveness against specific forms of network attack. This is very much an emerging area of research. Mapping from specific measures of system and network performance such as latency or throughput to their effect on overall mission success is another area in which there is not much work to compare to.

However, there is a extensive body of current and recent work on mapping tasks onto the resources available in a

*This work was supported by the United States Air Force and the Defense Advanced Research Projects Agency (DARPA), under contract from AFRL, contract # FA8750-11-C-0265. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited

[1] Amboro is a cloud forest in Bolivia.

distributed system, whether it is called a cloud, a grid, or just a network. Some of this work employs control-theoretic approaches to adaptive load-balancing for virtual machines across physical servers, in the presence of other resource constraints (Hyser et al. 2007; Wood et al. 2007), or optimizing the use of resources such as power, again under constraints derived from the tasks being supported (Wang and Wang 2010; Padal et al. 2007). Other research makes use of game theory and other methods for decision-making under uncertainty to allocate unreliable network assets to computational tasks (Sarmenta 2002; Sonnek, Chandra, and Weissman 2007), for example using statistics on past behavior to balance task throughput and reliability. Singh, Korupolu and Mohapatra (Singh, Korupolu, and Mohapatra 2008) describe a sophisticated online algorithm for hierarchical, multi-dimensional load-balancing across a distributed system, based on an algorithm for the multi-dimensional knapsack problem. In their conclusion, they discuss extending this work based on statistical information similar to that in the work cited just above.

What distinguishes our work is the need to represent and reason about expectations, trade offs involving risk and resource cost for deploying monitoring and active defenses, and the effect of network asset compromises on measures of mission effectiveness.

Finally, there are simulation-based tools that can be used to configure network resources, such as Opnet's IT Guru Network Planner [2]. Simulation and sampling approaches can be used to evaluate alternative network configurations, including the behavior of those configurations under adverse conditions such as network outages. These tools also provide or support static analysis tools, such as Opnet's "Survivability Report," and evaluation against regulatory policies and vendor "best practices." What they do not do is provide a means to find a good configuration in the first place, or to modify the current configuration as the situation changes.

CARINAE

The departure point for AMBORO is a system called Cyber Architecture Reasoner Inferring Network and Application Environments™ (CARINAE), described in (Michalowski, Boddy, and Carpenter 2010). As shown in Figure 1, CARINAE is a model-based, trust-driven tool for configuring defensive cyber operations. Given a network model and information regarding current and planned network operations in support of both missions and network defense, CARINAE provides network operators with the means to detect and resolve resource conflicts in network cyber-defense operations.

With CARINAE, network architects and operators can predict and resolve multiple scalability issues, including physical and logical network topologies, defended application resource loads, and defensive application architecture and resource requirements. Focused on large, service-oriented net-centric enterprise systems, CARINAE leverages constraint-based reasoning and open source, industry standard tools to create a robust analytical architecture that

[2]http://www.opnet.com/solutions/network_performance/index.html

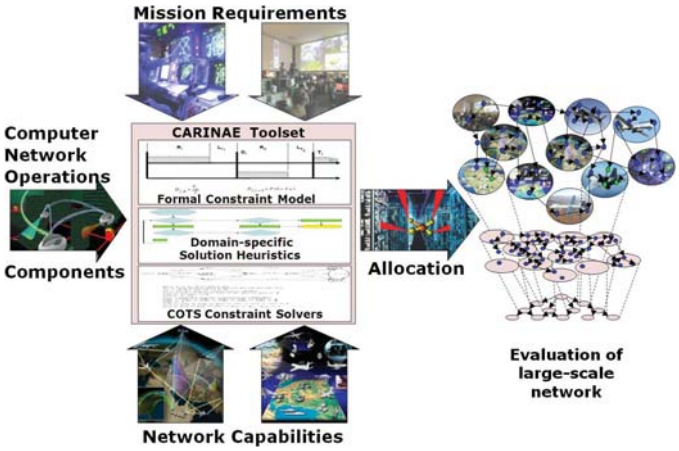


Figure 1: Cyber Architecture Reasoner Inferring Network and Application Environments™ (CARINAE)

can analyze the interactions between network configurations and mission requirements for large-scale defensive cyber applications (Michalowski, Boddy, and Carpenter 2010), (Haigh, Harp, and Payne 2010). CARINAE provides bandwidth, memory, and computational performance guarantees for large networks supporting diverse operational missions and defensive applications. Based on a collection of innovative modeling and algorithmic optimizations, CARINAE has been employed to analyze networks consisting of up to 1,000,000 nodes, with solution times typically well under a minute.

CARINAE's constraint-based model supports easy extension to a wide diversity of network topologies, node configurations, and mission requirements. The model supports the representation of hierarchies of processing nodes and hierarchical resource usage, supporting server-based virtualization and a hierarchy of network services, as well as network links tagged with various attributes, supporting requirements such as encrypted links. However, CARINAE does not support an explicit mapping from tasks to resource demands: the demands are provided directly. Nor does CARINAE's model or solution engines support reasoning about uncertainties such as the trustworthiness of a given node, or the likelihood of task or mission success.

In CARINAE, we employed the Coin-OR Linear Programming solver, as well as the Minizinc CSP solver. For AMBORO, we are currently using either the Minizinc default solver or the ECLiPse CSP solver. The AMBORO architecture is deliberately structured to make it easy to swap different solvers in and out as the computational and expressive needs of the system change.

AMBORO Architecture and Implementation

The process supported by AMBORO starts with the provision of a problem instance defined in the network and mission models. The mission model describes the tasks involved in the mission, including subtasks and any ordering or other temporal constraints, as well as computational

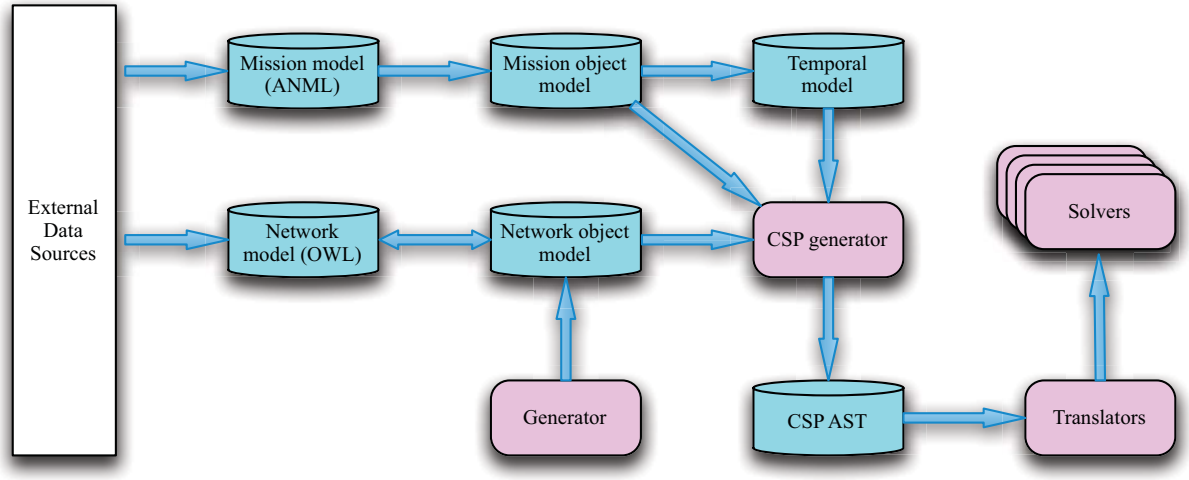


Figure 2: AMBORO Architecture

and communication requirements, which we will refer to as *demands*. The network model describes the computational and communication assets available, including their capacity limits and current configuration. Both models have textual representations in AMBORO, the mission model in the Action Notation Modeling Language (ANML), the network model in the Web Ontology Language (OWL). Both models are then parsed into internal Java data-structures, which are the primary representations operated on within AMBORO. The mission model is then analyzed by a separate module which extracts information about the temporal relationships between different activities, which is used to construct the multi-period schedule representing the progression of mission phases and activities.

The processing module labeled "CSP generator" then uses information from the network model, the mission model, and the temporal model to define an abstract syntax tree represented the desired configuration problem as a constrained optimization problem. This problem is then translated into the appropriate input language for one or more solving engines. At the current time, the solver output is being interpreted by hand. Integration with other Mission-oriented Resilient Clouds (MRC) capabilities will require that the solution format of whatever solver(s) are being employed be mapped back into the entities described in the configuration problem, as represented in the network and mission models.

At this point, we have a fully-functional, end-to-end implementation of AMBORO. Starting from a mission model and a network model, the system will extract the necessary constraints and formulate a multi-period CSP model covering multiple mission phases. This model is then automatically translated into minizinc and submitted to the solver, resulting in a feasible assignment or a notification of infeasibility.

The mission model includes assignable resources, multiple resources required for a given activity, and inter-activity resource constraints (e.g., you must use the same asset for two different activities). The mission model also supports the assignment of subsets of network assets, for example only allowing certain services to run on network nodes with a specified set of capabilities. The network model supports explicit representation of asset reliability, which is carried through the constraint model, all the way to an automatically-generated solution. *Over-provisioning* (the assignment of redundant resources to increase reliability) is currently modeled, but not yet part of the optimization problem.

AMBORO's optimization model has several additional capabilities, including optimizing for minimum distance from a previous assignment to processors (supporting incremental solutions), and a choice of network communication models, encompassing ignoring network liinks, requiring simple connectivity, or feasibility within specified throughput limits on links. Additionally, we are contemplating adding a term to the objective function that would encourage distributing processing demands across processors, as opposed to the natural aggregation that occurs if there is a finite probability of asset failure (because the probability of mission failure rises with the number of unreliable network assets being used).

Network Model

The AMBORO network model is very simple. Shown in Figure 3, the model includes a hierarchical network of *processing elements*, as well as a set of *links* among them. These are the elements that can be allocated as resources, which is all that is required.

There are certainly other aspects to the allocation problem involving the network, such as what demands are allowed on certain links or what services can be allocated to which processing elements. That information is kept in the mission model, which is where the demands are represented. There are additional constraints that represent the "physics"

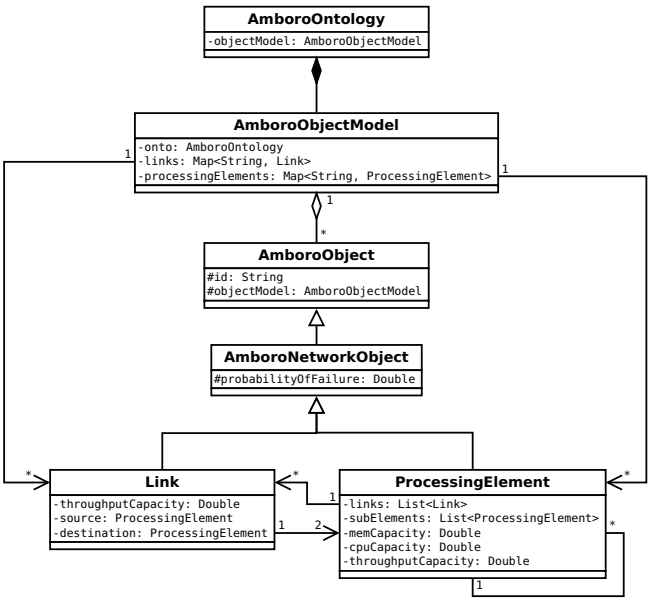


Figure 3: AMBORO Network model

of the network, for example how data flows from one node to another over the available links, or how computational resources used in a sub-element are aggregated into the containing element hierarchy (providing support for virtualization). These constraints are presented in the full constraint model, which space precludes including here.

As previously described, network model instances are stored and exchanged in OWL, which is inter-translated with the in-memory java object model for which Figure 3 provides the class model. The "Generator" function shown in Figure 2 can be used to automatically generate network model instances, which can then be used in the solution process, or back-translated into OWL for export to other tools, or for manual browsing.

Mission Model

Our mission models are represented in the Action Notation Modeling Language (ANML) (Smith, Frank, and Cushing 2008). As a domain modeling language for planning and scheduling applications, ANML provides support for building parameterized task models incorporating temporal constraints such as execution windows and minimum or maximum durations, resource requirements such as necessary tools or equipment, or capacity such as memory usage or network bandwidth. ANML provides a uniform semantics for both precondition/effect models of planning and task decomposition. A further advantage to ANML is that it is specifically designed to facilitate translation into constraint models, indeed has a semantics that is *defined* in terms of relationships represented as constraints. In this section, we present both the ANML representation of mission plans, and the ontology in which mission plans are represented internally.

Tactical Recovery of Aircraft and Personnel (TRAP) Planning Model

Figure 4 shows a simple TRAP mission task model, consisting of two main phases, corresponding to mission planning and execution. Arrows indicate precedence relationships on time points, showing, for example, that three different organizations must be involved in mission planning, all within the mission planning phase. Not shown graphically, but modeled in the ANML mission model presented below, is the requirement that all three of these planning activities happen at the same time. In the second phase, the relationships are different. Air support (AWACS and longer-range air cover provided by fighters) must be in place before close air support (provided by Apaches or other helicopters) can deploy, which must in turn be in place before any ground operations commence. Similarly for the mission return: each level of support must be withdrawn in order.

Additionally, these tasks all come with *resource requirements*. As shown in Figure 5, the planning phase requires local computing, storage, and network capabilities for each organization involved in the planning process, as well as a router that supports communication among them and back to the Global Information Grid (GIG) or other backend, large-scale data storage and processing resource. If any of these resources fail, then the planning phase cannot be completed, at least not until new resources are allocated. In Figure 5, the network has been configured such that there are redundant network paths among all the nodes. At least one of the routers must remain functional in order to maintain the required network connectivity.

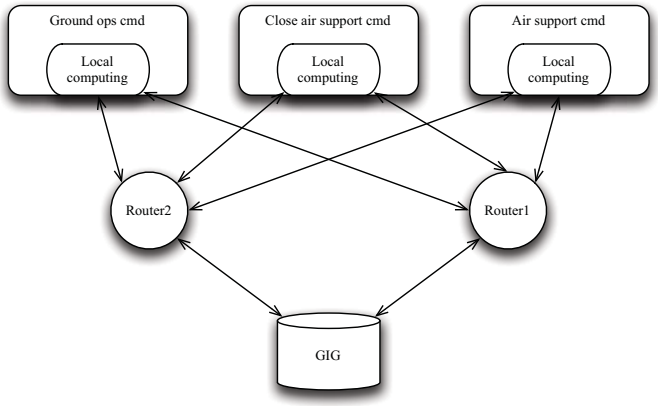


Figure 5: Network resources required for the planning phase.

Figure 6 shows a similar picture of network resources required for the operational phase of the mission. In this picture, all operational communication is routed through the Airborne Warning and Control System (AWACS) aircraft. Those communications include back to the various organizational commands, as well as to the various support aircraft involved. There is no communication link to the retrieval team, the assumption being that they are under some kind of communications restriction. One resource directly related to

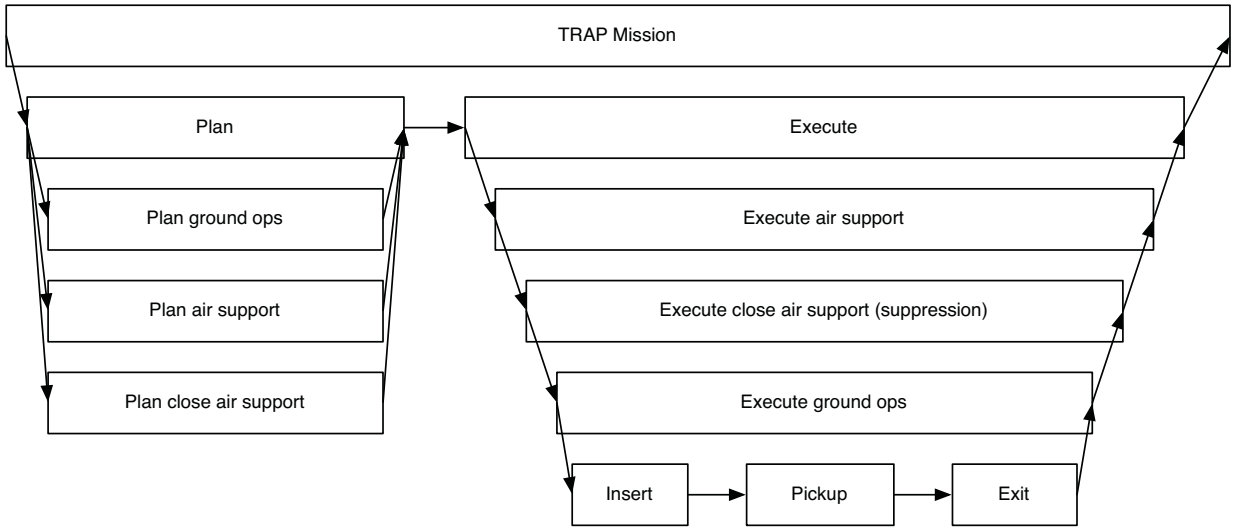


Figure 4: A simple TRAP mission plan

the retrieval team is their on-board computing resources. In particular, they will have pre-loaded map information and probably other data, in preference to have it sent during operational phase. Again, all of these resources must be functional, in order for the mission to succeed. This model can be complicated by adding the possibility of redundancy, by modeling decreased effectiveness rather than outright failure (e.g., do the mission anyway, but with potentially out-of-date imagery), and by modeling Byzantine failures.

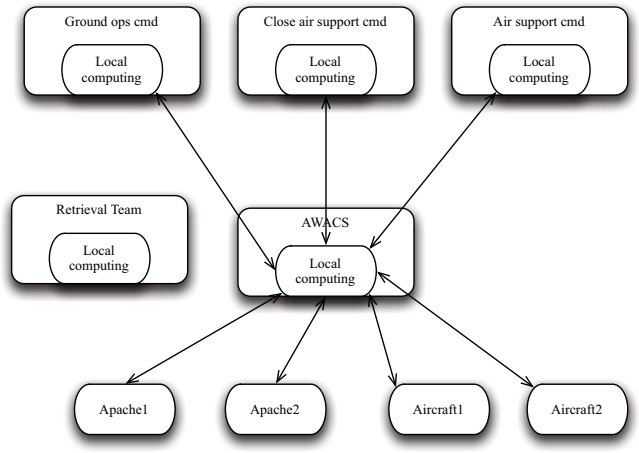


Figure 6: Network resources required for the operational phase.

ANML is a very expressive domain modeling language and could be used to represent a much more complex planning model, including for example alternative task decompositions, more complex constraints on resource assignments, or more general task parameters. The current AMBORO implementation supports more general resource assignments, but does not support alternative task breakdowns.

ANML Mission Planning Model

Here we present an ANML version of the planning model shown in Figure 4. First is the definition of the top-level action:

```
define action TRAP_mission() [duration]
{
  duration <= 360;  // 6-hour time limit
 [all] contains ordered(TRAP_plan(),
                        TRAP_execute());
}
```

This defines the top-level action as taking at most 6 hours, and requiring two subactions, `TRAP_plan` and `TRAP_execute`.

Next we define the planning action:

```
define action TRAP_plan() [duration]
{
  [all] contains {p1: TRAP_plan_ground();
                  p2: TRAP_plan_air();
                  p3: TRAP_plan_close();};

  start(p1) == start(p2) == start(p3);
  end(p1) == end(p2) == end(p3);

  [all] (status(Router1) == OK) |
        (status(Router2) == OK);
}
```

The planning action has three subactions, which are all constrained to start and end at the same times. In addition, it must be the case that either `Router1` or `Router2` is operational over the entire duration of `TRAP_plan`.

Here is one of the planning subactions:

```
define action TRAP_plan_ground()
{
  [all] status(ground_cmd_computing) == OK;
}
```

This very simple action contains a single condition, which is that the computing environment local to ground command is operational. We could also specify starting and ending times or a duration, but need not at this point: this model defines a sufficient set of constraints for a mission plan to be correct, rather than all of the constraints. The other three planning subactions are similar.

Next we define the execution subaction and one of its subactions:

```
define action TRAP_execute() [duration]
{
  [all] contains {e1: TRAP_execute_ground();
                  e2: TRAP_execute_air();
                  e3: TRAP_execute_close();};

  start(e1) <= start(e2) <= start(e3);
  end(e1) >= end(e2) >= end(e3);


  [all] status(AWACS_comms) == OK;
}

define action TRAP_execute_ground()
{
  [all] status(ground_cmd_computing) == OK;
}
```

The example presented here is simplified for clarity and brevity. AMBORO is capable of accepting models that include explicit resource assignments, for example making the decision to use `Router1`, `Router2`, or both an explicit part of the optimization.

Abstract Constraint Model

As described above, the CSP Generator builds an Abstract Syntax Tree (AST) constraint model from information contained in the network and mission models. To the extent practical, this model is abstracted away from the use of specific solver technologies (e.g., Constraint Logic Programming (CLP) versus Mixed-Integer Linear Programming (MILP)) and specific formulations (e.g., linear versus bilinear versus quadratic versus hybrid discrete/math models).

In this section, we present the current state of the abstract constraint model, including recent extensions to represent resource assignment to demands, and probability estimates for the failure of individual network assets. This is a formal specification of the model constructed by the "CSP generator" as shown in Figure 2. We start by defining an instance of a our CSP problem $\mathcal{C}$ as a tuple

$$\mathcal{C} = \langle \mathbf{E}, \mathbf{L}, \mathbf{D} \rangle$$

comprising

- a set $\mathbf{E}$ of *processing elements*,
- a set $\mathbf{L}$ of *links*, and
- a set $\mathbf{D}$ of *demands*.

In this section, we define each of these elements. Subsequent sections will discuss their interaction, and the constraints we add as a result. Constraints to be added to the model are defined in numbered equations.

All constraints in this model are expressed in time-free terms. AMBORO constructs a multi-period model by replicating the static model across periods, with inter-period links as needed for things like allocations to activities that span multiple periods. CPU demand is expressed as a rate requirement (e.g. MIPS). Communications demand is expressed as a requirement for a specified data rate. Memory demand is expressed as an amount of memory that must be allocated, out of a finite store.

Processing Elements

A processing element $e \in \mathbf{E}$ has the following attributes:

- a cpu capacity[3]: $\mathsf{cpu}(e) \mapsto \mathbb{R}^+$
- a memory capacity: $\mathsf{mem}(e) \mapsto \mathbb{R}^+$
- a set of sub-elements[4]: $\mathsf{sub}(e) \mapsto 2^{\mathbf{E}}$

Memory and CPU (and disk, if needed) are all sufficiently similar as currently modeled that, while we define the required attributes for all of these capacity resources, we only present the constraints for CPU. For now, all other capacity constraints local to processing elements look exactly the same as CPU.

A processing element also functions as a node in a network, defined by links, as described below. In addition to the throughput capacities defined on links, we also define a node throughput capacity associated with the processing element e:

$$\mathsf{rate}(e) \mapsto \mathbb{R}^+$$

Finally, a processing element is susceptible to failure. We define the probability that the processing element e will function correctly (i.e., provide the required cpu, memory, and networking capabilities):

$$\mathsf{prob}(e) \mapsto [0.0, 1.0]$$

Links

Communication connectivity between processing elements is provided by *links*. Links are directional, with the following attributes for a link $l \in \mathbf{L}$:

- a source processing element: $\mathsf{src}(l) \mapsto \mathbf{E}$
- a destination processing element: $\mathsf{dest}(l) \mapsto \mathbf{E}$
- throughput capacity: $\mathsf{rate}(l) \mapsto \mathbb{R}^+$

Demands

In this section, we address cpu and communication demands. The model and constraints for memory demands looks just like that for cpu demands. To differentiate between the different types of demands, we define subsets

[3] $\mathbb{R}^+$ denotes the set of non-negative real numbers. Any attribute denoting a value in $\mathbb{R}^+$ must be explicitly constrained to be ≥ 0, unless it's a constant.

[4] The symbol $2^{\mathbf{E}}$ denotes the *power set* of $\mathbf{E}$: the set of all subsets of $\mathbf{E}$, including $\mathbf{E}$ and the empty set $\emptyset$.

of $\mathbf{D}$: D_{cpu} and D_{comm}, each comprising all of the CPU and communication demands, respectively. CPU demands $d \in D_{cpu}$ have the following attributes:

- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$
- a set of processing elements to which the demand may be assigned: $\mathsf{allowed}_E(d) \in 2^{\mathbf{E}}$
- a variable [5] $\mathsf{orig}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_E(d)$

A communication demand $d \in \mathbf{D}_{comm}$ has

- a demand level: $\mathsf{demand}(d) \mapsto \mathbb{R}^+$
- a set of processing elements to which the source may be assigned: $\mathsf{allowed}_S(d) \in 2^{\mathbf{E}}$
- a set of processing elements to which the source may be assigned: $\mathsf{allowed}_D(d) \in 2^{\mathbf{E}}$
- a set of *allowed links*[6]: $\mathsf{allowed}_L(d) \mapsto 2^{\mathbf{L}}$
- a variable $\mathsf{src}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_S(d)$
- a variable $\mathsf{dest}(d)$, which will be assigned a value drawn from $\mathsf{allowed}_D(d)$

Processing Element Constraints

Processing elements are defined in a part/whole hierarchy of elements and sub-elements. For processing elements e_i, e_j, where $i \neq j$:

$$e_i \in \mathsf{sub}(e_j) \Rightarrow e_i \notin \mathsf{sub}(e_k), \forall k \neq j$$

and if we define $\prec$ as the transitive closure of the sub-element relation, then

$$e_i \prec e_j \Rightarrow e_j \not\prec e_i$$

These "constraints" are much more likely to be enforced as a property of the network model than to appear explicitly in the CSP to be solved.

There are two possible views of the processing element hierarchy. In the *aggregate* model, processing elements at any level in the hierarchy impose constraints corresponding to usage attributes, interpreted as resource limits to be compared to their aggregate-utilization attributes. For processing elements having no sub-elements, the aggregate-utilization attributes are set or solved for directly (see Subsection). For processing elements with sub-elements the aggregate-utilization attributes are computed from the aggregate-utilization attributes of their sub-elements. We define a variable:

- aggregate CPU utilization: $\mathsf{aggcpu}(e) \mapsto \mathbb{R}^+$

and add constraints:

$$\forall e \in \mathbf{E}, \mathsf{aggcpu}(e) \leq \mathsf{cpu}(e) \tag{1}$$

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggcpu}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{aggcpu}(e') \tag{2}$$

This is a good model for aggregated global resources such as power or comm. bandwidth, where at any level of the hierarchy the sum of the budgets for the next level down may be more than the capacity limit imposed (it is assumed that not everyone will draw their maximum budget at the same time).

In the *budget* model, processing element capacities impose constraints both down the hierarchy (resource limits) and up the hierarchy (resource demands). In this case, we replace the constraint 2 above with

$$\forall e \in \mathbf{E} : \mathsf{sub}(e) \neq \emptyset, \quad \mathsf{aggcpu}(e) = \sum_{e' \in \mathsf{sub}(e)} \mathsf{cpu}(e') \tag{3}$$

This is more appropriate for something like weight, or power budgets for sub-assemblies that don't get switched on and off. There is no requirement that either the aggregate or budget models be uniformly applied in a given processing element hierarchy; both may be needed, in different places.

CPU Demand Constraints

CPU and memory demands are imposed by constraining the corresponding aggregate utilization. For CPU[7]:

$$\forall e \in \mathbf{E}, \quad \mathsf{aggcpu}(e) = \sum_{d \in D_{cpu}} [\mathsf{orig}(d) = e] \, \mathsf{demand}(d) \tag{4}$$

Note that $\mathsf{sub}(\mathsf{orig}(d))$ must equal $\emptyset$ (i.e., the processing element to which d is applied may not have any sub-elements). This can be implicitly enforced via the membership of $\mathsf{allowed}_E(d)$.

Communication Demand Constraints

We can view the set of processing elements $\mathbf{E}$ and any set of links $L \subseteq \mathbf{L}$ in a given CSP model as a directed graph $G = \langle \mathbf{E}, L \rangle$, with vertices $\mathbf{E}$ and edges L, where each edge $l \in L$ is labeled with $\mathsf{rate}(l)$. Then G fits the definition of a specialized form of directed graph called a *flow network*.[8] Because different communication demands are represented as different flows, with distinct sources and sinks, we need to represent this as a *multi-commodity* flow problem, with each demand $d \in D_{comm}$ corresponding to a different commodity. We define the following with respect to a given set of links $L \subseteq \mathbf{L}$, with $l \in L$, $d \in D_{comm}$ and $e \in \mathbf{E}$:

The source and destination of a communication demand must be distinct:

$$\forall d \in D_{comm}, \quad src(d) \neq dest(d) \tag{5}$$

New variables:

[5] Arguably, variables should be defined in subsequent sections that define the optimization problem. We leave some of them as attributes of the different network entities because, depending on the model, a given attribute may move back and forth between being a variable and being a constant.

[6] The symbol $2^{\mathbf{L}}$ denotes the *power set* of $\mathbf{L}$: the set of all subsets of $\mathbf{L}$, including $\mathbf{L}$ and the empty set $\emptyset$.

[7] The notation used here is the Iversen bracket which is defined by $[S] = \begin{cases} 0 & \text{if } S \text{ is false} \\ 1 & \text{if } S \text{ is true} \end{cases}$

[8] `http://en.wikipedia.org/wiki/Flow_network`

- demand flow on a link: $\mathsf{flow}_L(l,d) \mapsto \mathbb{R}^+$
- demand inflow at a processing element: $\mathsf{inflow}_L(e,d) \mapsto \mathbb{R}^+$
- demand outflow at a processing element: $\mathsf{outflow}_L(e,d) \mapsto \mathbb{R}^+$

Demand inflow and outflow at a processing element is defined by the flow on the connected links:

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm}, \quad \mathsf{inflow}_L(e,d) = \sum_{l \in L} [\mathsf{dest}(l) = e]\, \mathsf{flow}_L(l,d) \tag{6}$$

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm}, \quad \mathsf{outflow}_L(e,d) = \sum_{l \in L} [\mathsf{src}(l) = e]\, \mathsf{flow}_L(l,d) \tag{7}$$

Demand can only flow on allowed links:

$$\forall l \in L, \forall d \in D_{comm} : l \notin \mathsf{allowed}_L(d), \quad \mathsf{flow}_L(l,d) = 0 \tag{8}$$

The following constraint enforces conservation of flow at each node in the network (each processing element) for each demand:

$$\forall e \in \mathbf{E}, \quad \forall d \in D_{comm} \quad \mathsf{inflow}_L(e,d) + [\mathsf{src}(d) = e]\, \mathsf{demand}(d) = \mathsf{outflow}_L(e,d) + [\mathsf{dest}(d) = e]\, \mathsf{demand}(d) \tag{9}$$

Note that according to this definition, there is no requirement that a given communication flow use a single path from one processing element to another. The throughput required may be spread over any or all of the possible paths between the two processing elements.

Note as well that representing dataflow as a material flow is a potentially-misleading simplification: data may be encrypted, filtered, decoded/expanded, or in other ways made larger or smaller at a given processing element, thus violating the conservation defined in this section and the next one. As long as the change in data size can be mapped to a change in required throughput, this can be added to the current model fairly easily.

Link and Node Rate Constraints

For convenience we define total flow on each link and processing element with respect to a given set of links $L \subseteq \mathbf{L}$, with $l \in L$ and $e \in \mathbf{E}$:

- total flow on a link: $\mathsf{flow}(L)l \mapsto \mathbb{R}^+$
- total inflow at a processing element: $\mathsf{inflow}_L(e) \mapsto \mathbb{R}^+$
- total outflow at a processing element: $\mathsf{outflow}_L(e) \mapsto \mathbb{R}^+$

The following constraints total up the flows across all demands for each link and processing element:

$$\forall l \in L, \quad \mathsf{flow}(L)l = \sum_{d \in D_{comm}} \mathsf{flow}(L)l, d \tag{10}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) = \sum_{d \in D_{comm}} \mathsf{inflow}_L(e,d) \tag{11}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{outflow}_L(e) = \sum_{d \in D_{comm}} \mathsf{outflow}_L(e,d) \tag{12}$$

Now we can specifiy the capacity constraints on links and processing elements:

$$\forall l \in L, \quad \mathsf{flow}(L)l \leq \mathsf{rate}(l) \tag{13}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) \leq \mathsf{rate}(e) \tag{14}$$

$$\forall e \in \mathbf{E}, \quad \mathsf{outflow}_L(e) \leq \mathsf{rate}(e) \tag{15}$$

For now we are assuming that flow for a processing element is constrained independently for in and out flow. But there is an alternative constraint that we could apply if we need to model a device that cannot send and recieve at the same time:

$$\forall e \in \mathbf{E}, \quad \mathsf{inflow}_L(e) + \mathsf{outflow}_L(e) \leq \mathsf{rate}(e) \tag{16}$$

Translating the Abstract Constraint Model

The final step before actually invoking a solver is to translate the problem instance into the appropriate input language. This process is accomplished by walking the Constraint Satisfaction Problem (CSP) AST, emitting the appropriate formulation. In the current implementation, the CSP AST is translated into MiniZinc, which can be used as input for either the default MiniZinc solver, or as input to Eclipse. As a way of showing the kinds of mappings that are required, FIgures 7 and 8 provide examples of the current translation from the formal model into MiniZinc.

There are quite a few more constraints in this model, including those defining how computational resources are aggregated within the element/sub-element hierarchy (i.e., how virtualization is modeled), and the constraints defining communication flows over the network, but this will provide a sense of the kinds of translation required.

This translation needs to be implemented in the inverse direction as well. Output format from many solvers, including MiniZinc, can be heavily tailored, but the variable assignments and costs must be mapped back into the entities in the mission and network models. This requires effectively inverting the two-step translation first from the network and mission models into the CSP AST, and then from there into the solver-specific code shown in this section.

Conclusion and Future Work

AMBORO's uniform, constraint-based representation for modeling mission tasks, cloud resources and the current set of assignments of resources to tasks provides several benefits, including easy extension of the model to reflect new types of missions, tasks, network components, and operational requirements such as QoS guarantees, as well as the presence of a wide variety of implemented solvers. Other advantages to a constraint-based approach include the ability to define partial solutions, culprit identification in the case of infeasibilities or other conflicts, incremental solutions as additional tasks are added, and the use of local search methods.

The work reported here is from the first six months of a planned four-year project. At this point, we have implemented and validated an end-to-end AMBORO system. To date, the network and mission models on which the system has been tested are small, and the probabilities from the which the likelihood of mission success is computed

```
%-----------------------------------------------------------------------------
% Processing Elements

% FORMAL: function cpu(E) -> R+
array [Elements] of float: cpu;
constraint assert(forall (e in Elements) (cpu[e] >= 0.0), "cpu must be >= 0.0");

% FORMAL: function sub(E) -> 2^E
array [Elements] of set of int: sub;

% FORMAL: function aggcpu(E) -> R+
array [Elements] of var float: aggcpu;
constraint forall (e in Elements) (aggcpu[e] >= 0.0);

% FORMAL: function rate(E) -> R+
array [Elements] of float: elementRate;
constraint assert(forall (e in Elements) (elementRate[e] >= 0.0), "rate must be >= 0.0");
```

Figure 7: Adding constraints on processing elements and network links

```
%-----------------------------------------------------------------------------
% Demands

% FORMAL: set D_cpu of CPU demands
int: nCpuDemands;
set of int: CpuDemands = 1..nCpuDemands;

% FORMAL: set D_comm of communication demands
int: nCommDemands;
set of int: CommDemands = 1..nCommDemands;

% FORMAL: function demand(D_cpu) -> R+
array [CpuDemands] of float: cpuDemand;
constraint assert(forall (d in CpuDemands) (cpuDemand[d] >= 0.0), "cpuDemand must be >= 0.0");

% FORMAL: function allowed_E(D_cpu) -> 2^E
array [CpuDemands] of set of int: cpuAllowedElements;

% FORMAL: function orig(D_cpu d) -> allowed_E(d)
% Implemented as an array of zero/one variables.
array [CpuDemands,Elements] of var {0,1}: cpuOrig;
constraint
  forall (d in CpuDemands) (
    % Only one element can be 1 for each demand.  This is the chosen
    % element to fill this demand.
    1 = sum(e in Elements)( cpuOrig[d,e] )
  );
constraint
  forall (d in CpuDemands) (
    forall (e in (Elements diff cpuAllowedElements[d])) (
      % Exclude elements that are not allowed to be used for this demand
      cpuOrig[d,e] = 0
    )
  );

constraint
  forall (d in CpuDemands) (
    forall (e in (Elements diff commAllowedSrcs[d])) (
      % Exclude elements that are not allowed to be src for this demand
      commSrc[d,e] = 0
    )
  );
```

Figure 8: Adding demand constraints

are largely independent. We are currently in the process of adding complexity and scale in all three areas: larger networks, more complex mission models, and more complex probability computations. Further along in the project, we will also be extending from the simple failure probabilities described here to a more complex (but not yet defined) notion of "compromised" assets, where different kinds of compromise have different, possibly stochastic effects on mission effectiveness.

All of these extensions pose potential computational challenges. The question of network scale we have already addressed: our previous work on CARINAE demonstrated an ability to scale to networks of up to a million nodes, with only limited optimizations in our formulation. Added complexity in the mission model may affect solving time in any of several ways. Adding additional mission phases or additional levels of sub-tasks to the current multi-period model increases the size of the problem linearly. Given the relatively weak coupling of adjacent periods, this should not have a great effect on computational effort. Additional inter-task constraints, unordered subtasks, or permitting alternative task decompositions are the extensions most likely to add difficulty to scaling up.

Moving to a more general and more complex stochastic model has the potential to add significant difficulties. The overall MRC program is currently in an early stage, thus we do not know exactly what additional complexities will be required. Depending on specific details for these extensions, our approach may range from off-line model simplification and exploitation of what independencies exist, using an existing probabilisitic modeling language such as Figaro (Pfeffer 2009), to numeric approximations exploiting the fact that prior failure probabilities tend to be very close to zero, to hybrid optimization methods involving partitioning a large, non-convex solution space, as for example in (Lamba et al. 2003).

References

Haigh, J.; Harp, S. A.; and Payne, C. N. 2010. Aimfirst: Planning for mission assurance. In *Proceedings of 5th International Conference on Information Warfare and Security*.

Hyser, C.; McKee, B.; Gardner, R.; and B.J.Watson. 2007. Autonomic virtual machine placement in the data center. Technical Report HPL-2007-189, HP Labs. `http://www.hpl.hp.com/techreports/2007/HPL-2007-189.pdf`.

Lamba, N.; Dietz, M.; Johnson, D. P.; and Boddy, M. 2003. A method for global optimization of large systems of quadratic constraints. In *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction*.

Michalowski, M.; Boddy, M.; and Carpenter, T. 2010. Coordinated management of large-scale networks using constraint satisfaction. In *Working Notes of the 2010 AAAI Workshop on Intelligent Security (SecArt)*.

Padal, P.; Shin, K. G.; Zhu, X.; Uysal, M.; Wang, Z.; Singhal, S.; Merchant, A.; and Salem, K. 2007. Adaptive control of virtualized resources in utility computing environments. In *EuroSys*.

Pfeffer, A. 2009. Figaro: An object-oriented probabilistic programming language. Technical report, Charles River Analytics.

Sarmenta, L. 2002. Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGrid*.

Singh, A.; Korupolu, M.; and Mohapatra, D. 2008. Server-storage virtualization: Integration and load balancing in data centers. In *Proceedings of the ACM/IEEE Conference on Supercomputing*.

Smith, D.; Frank, J.; and Cushing, W. 2008. The anml language. In *International Conference on Automated Planning and Scheduling*.

Sonnek, J.; Chandra, A.; and Weissman, J. 2007. Adaptive reputation-based scheduling on unreliable distributed infrastructures. In *TPDS*.

Wang, Y., and Wang, X. 2010. Power optimization with performance assurance for multi-tier applications in virtualized data centers. In *IEEE Online Conference on Green Computing*.

Wood, T.; Shenoy, P.; Venkataramani, A.; and Yousif, M. 2007. Black-box and gray-box strategies for virtual machine migration. In *In Proceedings of NSDI*.

Partially Grounded Planning as Quantified Boolean Formula

Michael Cashmore
University of Strathclyde
Glasgow, G1 1XH, UK
michael.cashmore@strath.ac.uk

Maria Fox
King's College London
London WC2R 2LS
maria.fox@kcl.ac.uk

Abstract

This paper introduces a new technique for translating bounded propositional reachability problems into Quantified Boolean Formulae (QBF) in a partially grounded manner. The approach uses the idea of domain-level lifting as an improvement to SAT without lifting. The technique is applicable to most SAT or QBF approaches as an additional improvement, potentially reducing the size of the resulting formula by an exponential amount. We present experimental results showing that the approach applied to a simple SAT translation greatly improves the time taken to encode and solve problems in which there are many objects of a single type, solving some problems that cannot be reasonably encoded as SAT.

1. Introduction

Planning as Satisfiability is one of the most well-known and effective techniques for classical planning: SATPLAN (Kautz and Selman 1992) was an award-winning system in the deterministic track for optimal planners in the first International Planning Competition (IPC) in 1998, the 4th IPC in 2004, and the 5th IPC in 2006. The basic idea is to encode the existence of a plan with $n + 1$ (or fewer) steps as a propositional (SAT) formula obtained by unfolding, n times, the symbolic transition relation of the automaton described by the planning problem. In recent work (Rintanen 2010) the basic SAT approach has been improved by equipping the solver with planning-specific variable and value-ordering heuristics that are similar to the helpful actions filter of FF (Hoffmann and Nebel 2001). This is very effective for solving planning problems in the SAT framework. In general, SAT-based planning, though quite successful, suffers from the problem that it is easy to come up with problems in which the number of steps required is large, making it impossible to even encode the original problem as a propositional formula. The same problem arises in bounded model checking (Biere et al. 1999). The use of compact encoding as Quantified Boolean Formulae (QBFs) combined with the use of QBF solvers has been proposed (Jussila and Biere 2007; Mangassarian, Veneris, and Benedetti 2010; Cashmore and Fox 2010; Dershowitz, Hanna, and Katz 2005) as a way to overcome this problem. In particular, Rintanen (2001), Jussila and Biere (2007) and Cashmore and Fox (2010) present encodings that are logarithmic in n, resembling the proof of the PSPACE-hardness of solving QBFs (Savitch 1970; Stockmeyer and Meyer 1973).

Here we introduce a technique for translating bounded propositional planning problems into Quantified Boolean Formulae (QBF). The technique differs from the previous in that it uses the idea of domain-level lifting of objects into equivalence classes. This enables the construction of *partially grounded* boolean formulae.

In many planning domains there exist multiple objects of the same type. For example in the *driverlog* domain there might be many trucks and drivers. In order to encode these problems into a SAT formula the planning instance is first grounded, to create all possible propositions using different truck and driver combinations. This involves uniquely describing each object, and the relationships between them - i.e. in *driverlog* this means that the $board(driver, truck)$ action is copied for each driver/truck combination. Our approach describes only a single object, eg: "truck", which represents the equivalence class of trucks. The idea supports least-commitment planning (Tate 1976; Carbonell et al. 1990; Penberthy and Weld 1992), in which no variables are instantiated until there remains no choice about how they can be bound.

There is growing interest in the planning community (Nguyen and Kambhampati 2001; Ridder and Fox 2011) in the question of how to avoid grounding, and the work described here shows how grounding can be avoided in SAT. It is clear that there is the potential for lifted encodings to be exponentially smaller than grounded ones. Of course, an instance will usually require to be partially grounded in order to ensure that relationships between objects are properly preserved, so it is not always possible to achieve exponential improvement.

Our technique exploits the PSPACE complexity of the QBF problem to create encodings which are partially grounded. The resulting formula can require exponentially fewer variables (and clauses) than a corresponding SAT encoding to describe the state (and transition relation). The potentially exponential reduction is a function of the objects in the domain and is greatest in domains in which there are many objects of a single type. In order to determine the effectiveness of the encoding we run experiments on selected domains showing that the QBF encoding scales better as the number of objects, and size of the instance, increases. In

addition we show that there exist problems which can be solved by the QBF translation that remain unsolved by SAT.

We briefly describe the QBF and Planning problems in Section 2. We describe the QBF encoding in detail in Section 3. and provide an example in Section 4. Section 5. will detail the experiments run on the encoding and discuss the results. Finally we discuss some related work and conclude in Sections 6. and 7.

2. Preliminaries

Quantified Boolean Formula

Formally, the language of QBFs extends propositional logic by allowing for universal ($\forall$) and existential ($\exists$) quantification over variables (in our case, fluents and actions). Semantically, $\forall x.\varphi$ (resp. $\exists x.\varphi$) can be interpreted as $(\varphi_x \wedge \varphi_{\neg x})$ (resp. $(\varphi_x \vee \varphi_{\neg x})$), where φ_x (resp. $\varphi_{\neg x}$) is the formula obtained from φ by replacing x with $\top$ (resp. $\bot$).[1] The process of substituting $\forall x.\varphi$ (resp. $\exists x.\varphi$) with $(\varphi_x \wedge \varphi_{\neg x})$ (resp. $(\varphi_x \vee \varphi_{\neg x})$) is called *expansion*. By expanding all quantifiers, each QBF can be reduced to (possibly an exponentially larger) propositional formula. When every variable is expanded (in which case we say the QBF is *closed*), such an expansion reduces to a Boolean combination of $\top$ and $\bot$ and is thus equivalent to either $\top$ or $\bot$. The QBF formula expands recursively into a tree-structured representation where all the propositional variables are at the leaves. Expansion therefore produces the conjunctive binary tree that accords to the semantics of the QBF. Note that SAT is a subproblem of QBF in which every variables is implicitly quantified existentially ($\exists$).

The Planning Problem

Although the idea of partially grounding with QBF can be applied to any representation, Planning problems here are specified using the standard STRIPS formulation (Fikes and Nilsson 1971). The world is described with F, a set of *fluents*. An assignment of true or false to these fluents describes a state. These fluents are generated from the propositions of the domain and the objects in the problem instance through the process of *grounding*. Grounding is described in detail below. An *n-ary* proposition is represented as shown by figure 1.

I denotes the initial state of the world. The goal state is described by G, a formula over F.

The world is changed through the use of *operators*. An *n-ary* operator will be represented as shown by figure 2. Grounding these operators generates the set A of *action fluents*.

Traditionally, in order to translate the planning problem into boolean encodings we must first ground the instance. Grounding means generating fluents and action fluents from the operators and propositions of the domain. Briefly, the set of fluents are found by making every possible valid binding of objects to propositions. Action fluents are similarly created from the operators. For example, the operator defined by figure 2 would be grounded into $(|P||H|)$ action fluents, where $|P|$ is the number of objects of type $pigeon$ and $|H|$ is the number of objects of type $pigeonhole$. Each of these action fluents represents a different $pigeon$/$pigeonhole$ pairing.

The encoding described in Section 3. is only partially grounded. This means that some operators will not be grounded at all, or only some parameters will be bound, creating a set of partially grounded operators. These operators will be represented by variables in the QBF encoding.

Planning as SAT

Consider a planning problem $\Pi = \langle F, A, I, G\rangle$. As standard in planning as satisfiability, the existence of a parallel plan with makespan n is proved by building a propositional formula with n copies of the sets F and A. In the following,

- by X_α we denote one such copy of the set of variables;
- by $I(X_\alpha)$ (resp. $G(X_\alpha)$) we denote the formula obtained from I (resp. G) by substituting each $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$;
- by $\sigma(X_\alpha)$ we denote the formula obtained from σ by substituting each variable $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$;
- by $\tau(X_\alpha, X_\beta)$ we denote the formula obtained from τ by substituting each variable $x \in X$ with the corresponding variable $x_\alpha \in X_\alpha$ and similarly each $x' \in X'$ with the corresponding $x_\beta \in X_\beta$.

The state constraint $\sigma(X)$ is defined as:

- each action fluent in X implies a conjunction of fluents (in X) corresponding to its preconditions;
- each pair of action fluents in X that are *mutually exclusive* form a binary disjunction of their negations.

The transition relation $\tau(X, X')$ is defined as:

- each action fluent in X implies its effects in X';
- each fluent in X' implies a disjunction of supporting actions and itself in X, (explanatory frame axioms).

For $n \geq 1$, the *planning problem* Π *with makespan* n is the Boolean formula Π_n defined as

$$I(X_1) \wedge \bigwedge_{i=1}^{n} \sigma(X_i) \wedge \bigwedge_{i=1}^{n} \tau(X_i, X_{i+1}) \wedge G(X_{n+1}) \quad (1)$$

and a *plan for* Π_n is an interpretation satisfying (1).

However, since the plan existence problem (assuming, as we do, a deterministic transition relation and a single initial state) is a PSPACE-complete problem (Bylander 1994) the size of (1) can be exponential in the number of fluents - making it impossible to even build (1). QBFs are a promising alternative representation language given that:

1. there exist encodings of the planning problem with makespan n as QBFs which are polynomial in the number of fluents, and
2. there is a growing interest in developing efficient solvers for QBFs; see, for example, the report from the 2010 QBF competition (Peschiera et al. 2010).

[1] $\top$ and $\bot$ are the logical symbols we use for truth and falsity.

$$IN\,(?p - pigeon,\, ?h - pigeonhole)$$

Figure 1: Proposition representation for the proposition *in* in the pigeonhole domain.

$$\begin{array}{l} PLACE\,(?p - pigeon,\, ?h - pigeonhole) \\ \quad PRE:\, \neg PLACED(p), \quad EMPTY(h) \\ \quad EFF:\, PLACED(p), \quad \neg EMPTY(h), \quad IN(p,\, h) \end{array}$$

Figure 2: Operator representation for the operator *place* in the pigeonhole domain.

3. Partially Grounded QBF Encoding

For $n \geq 1$, the *planning problem* Π *with makespan* n is the Quantified Boolean formula Φ_n defined as

$$\begin{array}{l} \exists X_1^g \ldots X_{n+1}^g \forall a_1 \ldots a_m \exists X_1^u \ldots X_{n+1}^u \\ \bigwedge_{i=1}^{n} \tau_{qbf}(X_i^g \cup X_i^u, X_{i+1}^g \cup X_{i+1}^u) \\ \wedge \bigwedge_{i=1}^{n} \sigma_{qbf}(X_i^g \cup X_i^u) \\ \wedge I(X_1^g \cup X_1^u) \wedge G(X_{n+1}^g \cup X_{n+1}^u) \end{array} \tag{2}$$

and a *plan for* Φ_n is an interpretation satisfying (2).

$X_i^g \cup X_i^u$ represents the state of the Planning problem at timestep i. X_i^g denotes the grounded portion of the state, while X_i^u the ungrounded portion. σ_{qbf} and τ_{qbf} are analogous to σ and τ, and are similar in construction. σ_{qbf} and τ_{qbf} contain additional constraints to ensure consistency. These constraints are described later.

The introduction and then expansion of the universal variables, $a_1 \ldots a_m$, creates 2^m subformulae, which are conjunctively combined. The grounded portion of the problem is existentially quantified before the universals. The ungrounded variables are quantified (existentially) after these universals. Intuitively, each subformula represents a single object of the ungrounded type and the conjunction of these formulae represents an equivalent grounded plan. This is illustrated in figure 3.

Separating the State

Kautz and Selman (1996) introduced the idea of *simple operator splitting*, which replaces each n-ary operator with n unary operators throughout the encoding. For example, recall the operator in figure 2. In an ordinary SAT representation that uses action fluents this operator would be grounded. This would produce $(|P||H|)$ action fluent variables per state in the resulting formula. This operator, which contains two parameters, will be replaced by two unary split operators, namely; $PLACE_1(?p)$ and $PLACE_2(?h)$. This reduces the number of grounded actions from $(|P||H|)$ to $(|P| + |H|)$ per state. The partially grounded problem uses this idea to separate both operators and propositions before encoding as QBF.

X^g and X^u represent the grounded and ungrounded portions of the state respectively. All split operators, or split propositions that involve an object to remain ungrounded are represented by a variable added to X^u. All other propositions and operators are grounded, as described in Section 2. and added to X^g. n-ary operators which contain only ungrounded parameters are split into $n + 1$ parts. The extra split operator variable represents whether the operator is performed, and is added to X^g.

Ordinarily an action fluent $PLACE_1(?p)$ would be included in the formula for each pigeon object. In this case only a single variable is introduced and represents this operator for all pigeons. This reduces the number of variables representing this operator from $(|P| + |H|)$ to $(1 + |H|)$, in the case where only $pigeon$ objects are not grounded, and three in the case where $pigeonhole$ objects are also not grounded.

It might seem surprising that the $(|P||H|)$ grounded action fluents for $PLACE$ can be described using only three variables. However, the number of assignments made to these variables depends upon the number of universal variables quantified before these. For example, $a_1 \ldots a_m$ are expanded to create 2^m subformulae. The conjuction of these contains 2^m copies of $PLACE_1(?p)$, each of which pertains to a different $pigeon$ object. In addition a further $(log(|P||H|))$ existential variables will be added to ensure consistency. This is explained in more detail below.

Ensuring Consistency Between Leaves

Additional existential variables and constraints must be added to the problem in order to ensure plan validity. These constraints ensure that only a single object is bound to each parameter of an operator and that two actions that are mutually exclusive, but described in different leaves of the expanded QBF tree, cannot be performed in the same state. They also ensure split propositions that form one whole proposition, but are described in different leaves of the QBF tree, are bound to one another. That is:

1. if a split operator variable is made true, *exactly* one variable representing the other parameters of the operator must also be made true. This means that, if the parameter is grounded, one action fluent for this parameter must be made true in X^g. If ungrounded, the split operator variable for this parameter must be made true in *exactly* one universal branch of the QBF;
2. if one split operator variable is made true in X^u, the variable must be made false in every other branch of the QBF;
3. if one split operator variable is made true in X^u, all of its preconditions and effects that reside in other branches of the QBF must also be made true; and
4. if one split proposition variable is made true in X'^u, and is not supported by an action in the previous state, X^u, then

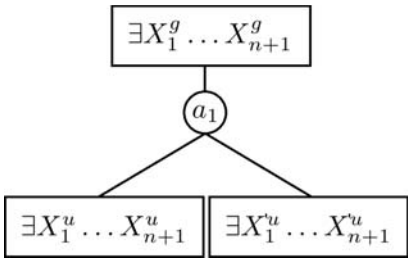


Figure 3: Φ_n for $m = 1$. The expansion of a_1 creates two subformulae, $\Phi_{n,\neg a_1}$ and Φ_{n,a_1} , which are conjunctively combined. Variables $X_1^g \dots X_{n+1}^g$ appear in both subformulae. Each branch of the expansion, $X_1^u \dots X_{n+1}^u$ and $X_1'^u \dots X_{n+1}'^u$, appears in a different subformula. Each subformula will correspond to the plan for a unique object.

it describes part of the same fluent as the corresponding split proposition variable in X^u.

These constraints are described more formally alongside an example below. The new variables will be referred to as a *mutex lock* and are constrained to be equal to the universal variables that define which branch (and so which unique object) is bound to the parameter.

For each split operator $o_u \in X^u$ a mutex lock is added to X^g of the form $lock_0^{o_u} \dots lock_m^{o_u}$. For each ungrounded split proposition $p^u \in X^u$ a mutex lock is added for each *other* ungrounded parameter of the whole proposition. The lock is of the form $lock_0^{p_u} \dots lock_m^{p_u}$.

The example we use is the pigeonhole domain in which both $pigeon$ and $pigeonhole$ objects are ungrounded. In addition to the proposition *in* and operator *place* the domain includes the propositions *placed* and *empty*, described in figure 4.

In the pigeonhole domain:

$$PLACE_0 \in X^g$$

and

$$PLACE_1, PLACE_2,$$
$$IN_1, IN_2, EMPTY, PLACED \in X^u$$

In addition locks are required for both split operators $PLACE_1$ and $PLACE_2$ and for the split propositions IN_1 and IN_2:

$$lock_1^{PLACE_1} \dots lock_m^{PLACE_1},$$
$$lock_1^{PLACE_2} \dots lock_m^{PLACE_2} \in X^g$$

and

$$lock_1^{IN_1} \dots lock_m^{IN_1},$$
$$lock_1^{IN_2} \dots lock_m^{IN_2} \in X^u$$

where $m = log(max(|P||H|))$. Thus $|X| = 7 + 4m$. Increasing the number of $pigeons$ and $pigeonholes$ increases m and $|X|$ logarithmically.

An operator O is split into O^g and O^u –grounded split operator variables and ungrounded, respectively. The constraints (1) and (2) extend $\sigma_{qbf}(X)$ for each $o_g \in O^g$ to contain:

- $\bigvee_{i=1}^{|o_g|} w_i^{o_g} \wedge \bigwedge_{i=1}^{m}(a_i \leftrightarrow lock_i^{o_u}) \rightarrow o_u, \forall o_u \in O^u$

and for each $o_u \in O^u$:

- $o_u \rightarrow \bigwedge_{i=1}^{m}(a_i \leftrightarrow lock_i^{o_u})$
- $o_u \rightarrow \bigvee_{i=1}^{|o_g|} w_i^{o_g}, \forall o_g \in O^g$

where $|o_g|$ is the number of action fluents generated by grounding split operator o_g and $w_i^{o_g}$ is the i^{th} such action fluent. In the pigeonhole domain this looks like:

- $PLACE_0 \wedge \bigwedge_{i=1}^{m}(a_i \leftrightarrow lock_i^{PLACE_j}) \rightarrow PLACE_j$ for $j = 1, 2$
- $PLACE_j \rightarrow \bigwedge_{i=1}^{m}(a_i \leftrightarrow lock_i^{PLACE_j})$, for $j = 1, 2$
- $PLACE_j \rightarrow PLACE_0$, for $j = 1, 2$

For each $o_u \in O^u$, $\sigma_{qbf}(X)$ is extended as described by constraint (3) to ensure correct preconditions:

- $o_u \rightarrow (lock_i^{p_u} \leftrightarrow lock_i^{\hat{o}_u})$ for each split proposition p_u in the preconditions of o_u with multiple ungrounded parameters and each ungrounded split proposition $\hat{p}_u$ associated with p_u.

where $\hat{o}_u$ refers to the split operator that must bind to the same object as the split proposition $\hat{p}_u$. To ensure correct effects $\tau_{qbf}(X, X')$ is extended to contain:

- $o_u \rightarrow (lock_j'^{p_u} \leftrightarrow lock_j^{\hat{o}_u})$ for $j = 1 \dots m$ for each split proposition p_u in the effects of o_u with multiple ungrounded parameters and each ungrounded split proposition $\hat{p}_u$ associated with p_u.

where $\hat{o}_u$ refers to the split operator that binds to the same object as the split proposition $\hat{p}_u$. Here $o_u, lock_j^{\hat{o}_u}, lock_j^{p_u} \in X$ and $lock_j'^{p_u} \in X'$. In the pigeonhole domain this takes the form[2]:

- $PLACE_1 \rightarrow (lock_j'^{IN_1} \leftrightarrow lock_j^{PLACE_2})$ for $j = 1 \dots m$
- $PLACE_2 \rightarrow (lock_j'^{IN_2} \leftrightarrow lock_j^{PLACE_1})$ for $j = 1 \dots m$

This ensures that the lock associated with $IN_2(?h)$ is bound to the same pigeon as the split operator $PLACE_1(?p)$.

Constraint (4) extends $\tau_{qbf}(X, X')$ in the following way for each $p_u \in X^u$:

[2]note that the operator $PLACE$ does not contain any split propositions in its preconditions and so only requires one of the constraints described by (3).

$EMPTY\,(?h - pigeonhole)$
$PLACED\,(?p - pigeon)$

Figure 4: Propositions *placed* and *empty* in the pigeonhole domain.

- $p_u \rightarrow \bigwedge_{i=1}^{m}(lock_i^{p_u} \leftrightarrow lock_i'^{p_u}) \vee supp(p_u)$ for each mutex lock of p_u.

where $supp(p_u)$ refers to a disjunction of supporting actions for p_u. In the pigeonhole domain this looks like:

- $IN_j' \rightarrow \bigwedge_{i=1}^{m}(lock_i'^{IN_j} \leftrightarrow lock_i^{IN_j}) \vee PLACE_j$ for $j = 1, 2$

Additional constraints that do not include mutex locks enforce preconditions and effects in the same way as the corresponding SAT encoding. These are described fully in Section 4.

If it is possible to use multiple copies of O in a single timestep, identical operators are included –with accompanying locks. The number of copied operators required is $min(|o|)$, $o \in O^g \cup O^u$. For example, in the pigeonhole domain this generates $3min(|P|, |H|) + 2log(|P||H|)$ variables, rather than the $(|P||H|)$ actions that would be generated from grounding.

4. Example

Here we describe the partially grounded QBF representation Φ_n of the pigeonhole domain in which both $pigeon$ and $pigeonhole$ objects are ungrounded. Φ_n, X^g and X^u are as described in Section 3. The goal is to have all $pigeons$ placed.

$\sigma_{qbf}(X)$ is the formula over X that includes the constraints:

- $PLACE_1 \rightarrow \neg PLACED$
- $PLACE_2 \rightarrow EMPTY$
- $PLACE_j \rightarrow (a_i \leftrightarrow lock_i^{PLACE_j})$ for $i = 1 \ldots m$, and $j = 1, 2$
- $PLACE_j \rightarrow PLACE_0$, for $j = 1, 2$
- $PLACE_0 \wedge \bigwedge_{i=1}^{m}(a_i \leftrightarrow lock_i^{PLACE_j}) \rightarrow PLACE_j$, for $j = 1, 2$

and $\tau_{qbf}(X, X')$ is the formula over $X \cup X'$ that includes the constraints:

- $EMPTY' \rightarrow EMPTY$
- $\neg EMPTY' \rightarrow \neg EMPTY \vee PLACE_2$
- $PLACED' \rightarrow PLACED \vee PLACE_1$
- $\neg PLACED' \rightarrow \neg PLACED$
- $\neg IN_j' \rightarrow \neg IN_j$, for $j = 1, 2$
- $IN_j' \rightarrow IN_j \vee PLACE_j$, for $j = 1, 2$
- $IN_j' \rightarrow \bigwedge_{i=1}^{m}(lock_i'^{IN_j} \leftrightarrow lock_i^{IN_j}) \vee PLACE_j$, for $j = 1, 2$

to support facts (explanatory frame axioms) and:

- $PLACE_1 \rightarrow PLACED'$
- $PLACE_2 \rightarrow \neg EMPTY'$
- $PLACE_1 \rightarrow (lock_j'^{IN_1} \leftrightarrow lock_j^{PLACE_2})$, for $j = 1 \ldots m$
- $PLACE_2 \rightarrow (lock_j'^{IN_2} \leftrightarrow lock_j^{PLACE_1})$, for $j = 1 \ldots m$

where $PLACED', EMPTY', IN_j', lock_y'^{x} \in X'$, to enforce operator effects.

$I(X)$ is defined by:

$$\neg PLACED \wedge EMPTY \wedge \neg IN_1 \wedge \neg IN_2$$

and $G(X)$ by:

$$PLACED$$

In the case where $|P| \neq |H|$ it is possible to constrain some operators to remain false, effectively excluding those objects from the plan. For example suppose $|P| = 2|H|$, the constraint

$$a1 \rightarrow \neg PLACE_2$$

is added to $\sigma_{qbf}(X)$.

5. Results

Experiments were run on several domains to determine the effectiveness of the encoding. We hypothesised that:

- as the size of the problem increased, the partially grounded QBF approach would scale better than the SAT approach in both encoding time and solving time;
- as the size of the problem increased, partially grounded QBF would find solutions faster than the SAT approach;
- we would find problems that were too large to encode in SAT within the time limit allowed, but that could be encoded and solved using partially grounded QBF.

The domains selected for experimentation were the *pigeonhole*, *gripper* and *blocksworld* domains. These domains were chosen as they work well with ungrounded approaches. In other domains in which there is very little or no benefit from lifting the partially grounded QBF encoding resembles the SAT encoding.

For each domain a number of problems were generated, gradually increasing the size of the domain. These problems were then translated into SAT and partially grounded QBF (PG-QBF) encodings. The time taken for this translation was recorded. We used the SAT encoding used by SATPLAN'06 (2006) as it used the same STRIPS-based fluent/action representation as the PG-QBF encoding. Other SAT encodings, and additional constraints can also be used as a basis for partially grounded QBF encodings.

The encodings were then solved using the SAT solver *picosat-535* and the QBF solver *quantor-3.0*. Descriptions

of these solvers can be found in Biere (2004) and (2008) respectively.

The times are recorded in Table 1. Times are presented in seconds, a hyphen indicating that the process was terminated after the time limit of 2 hours was reached. A star indicates that the process ran out of memory before the time limit was reached. The experiments were all run on a machine with 8GB of RAM and no artificial bound on the amount of memory used.

As can be observed, PG-QBF was able to encode and solve all of the pigeonhole instances considered, within the 2 hour limit, while SAT could not encode pigeonhole64, and was unable to solve pigeonhole32 and pigeonhole64 in the time available. PG-QBF was able to encode all of the instances across all domains in under one third of a second, while the time required by SAT to encode large pigeonhole instances grew exponentially. Both approaches exhibit exponentially growing solution time, although the PG-QBF curve increases more slowly than the SAT curve. Neither approach was able to solve gripper32. These results support our first and second hypotheses, that PG-QBF scales better than SAT in both encoding and solution time, and that PG-QBF solves problems faster than SAT. Our third hypothesis is supported by pigeonhole64 and blocksworld32, which demonstrates that there are indeed instances that cannot be encoded by SAT, but can be encoded by PG-QBF, within a fixed time limit.

6. Related Work

The partially grounded QBF encoding builds on the ideas of least-commitment planning and has similarities with symmetry-breaking, however, there are important differences between these techniques.

Planning with ungrounded operators in least-commitment planning (Tate 1976; Carbonell et al. 1990; Penberthy and Weld 1992) avoids explicit construction of the whole search space of the problem, but does not prune away unnecessary portions of the search space. Performing an operator in least-commitment planning makes true the disjunction of all the ground action fluents of that operator, implying that at least one must have been performed. However, this is not the case in partially grounded QBF. Any assignment to variables that represent an ungrounded operator in the QBF encoding will either make true or false that a ground, or set of ground action fluents have been performed. That is, a conjunction of action fluents is made either true or false. It is not possible to make an assignment that creates a disjunction. We are hoping to prune more of the search space away with each decision, rather than less, and so benefit from the lightweight encoding.

Fox and Long (1999) explain how symmetries can be discovered and exploited in planning problems in order to reduce search times. They use the *gripper* domain as an example where symmetry based techniques have a powerful effect, and indeed this is one of the domains used in our experimentation. Their planner STAN is a Graphplan-based planner that uses the analysis and exploitation of symmetry. It does this by avoiding considering symmetric alternatives after backtracking. For example, if all $pigeonholes$ were found to be symmetric and no plan could be found by graph layer n after placing $pigeon_0$ into $pigeonhole_0$, then no plan will be found after placing $pigeon_0$ into any other $pigeonhole_i$.

Although PG-QBF achieves a similar effect of avoiding search over equivalent ground instances of a problem, we do not spend any time in identifying symmetric relationships and we describe two objects of the same type using the same operators and propositions even if their initial states, goal states and entire plan trajectories differ. The advantage that we gain is that the lifting, and all the machinery required to ensure consistent handling of objects of the same type, is done entirely in the encoding of the problem and the approach is completely solver-independent. By contrast, STAN implements specialised techniques for backtracking when symmetry is present, and their detection of symmetry cannot be autimatically exploited by other planners.

Our approach is similar to the idea described by Rintanen (2003) which introduces new constraints into a problem encoding that disallows symmetric choices. In particular Rintanen's work shows how to avoid repeating symmetric mistakes made over more than one transition, and the mechanisms required to achieve this are part of the encoding rather than the solver. Our approach differs in that we do not encode symmetry-breaking constraints which can have the effect of greatly increasing the sizes of problem encodings. Also, our approach is applicable to domains in which there are no exploitable symmetries (eg: *blocksworld*), and can provide massive benefits as demonstrated by our results.

7. Conclusions

In this paper we have introduced a method for lifting SAT encodings of planning problems into Quantified Boolean formulae. Lifting is a very powerful idea, being explored in different ways by a number of researchers in planning. It is well-known that grounding of planning domains is infeasible for very large problems and that techniques for lifting planning instances can help with the solution of very large instances containing very large numbers of objects of the same type.

Our approach shows how to construct QBF encodings that both lift sets of similar objects into representative variables, and ensure consistent reasoning when the specific objects involved in transitions are not yet committed to. The approach is inspired by earlier work in least-commitment planning.

We have shown that our approach can lead to exponentially smaller encodings and allow larger problem instances to be solved than is possible using SAT.

References

Biere, A.; Cimatti, A.; Clarke, E.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, 193–207.

Biere, A. 2004. Resolve and expand. In *Proc. SAT*, 59–70.

Biere, A. 2008. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*.

	SAT			PG-QBF		
problem	encoding	solving	size	encoding	solving	size
pigeonhole2	0.04	0.00	0.89	0.04	0.00	1.04
pigeonhole4	0.09	0.00	12.60	0.04	0.00	2.65
pigeonhole8	0.33	0.12	292.82	0.06	0.00	7.13
pigeonhole16	5.53	5.06	8574.06	0.1	0.09	18.28
pigeonhole32	155.71	*	279783.22	0.13	0.58	43.67
pigeonhole64	-	-	-	0.16	22.8	111.82
gripper2	0.06	0.00	6.44	0.07	0.00	2.61
gripper4	0.13	0.03	36.06	0.1	0.01	7.52
gripper8	0.28	6.61	215.63	0.12	0.35	20.54
gripper16	1.22	1171.49	1492.95	0.16	180.19	49.92
gripper32	7.03	-	11137.96	0.3	-	123.68
blocksworld2	0.05	0.00	2.51	0.04	0.00	4.01
blocksworld4	0.11	0.01	44.50	0.09	0.02	17.36
blocksworld8	0.93	0.56	1118.58	0.13	0.16	52.36
blocksworld16	13.43	20.47	35277.29	0.18	1.16	152.26
blocksworld32	-	-	-	0.32	5834.88	399.88
blocksworld64	-	-	-	0.68	-	986.62

Table 1: Time taken to encode and solve problems, and problem sizes, using partially grounded QBF encodings and SAT based encodings. All times are in seconds, sizes in KB.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artif. Intell.* 69(1-2):165–204.

Carbonell, J.; Etzioni, O.; Gil, A.; Joseph, R.; Knoblock, C.; Minton, S.; and Veloso, M. 1990. Prodigy: An integrated architecture for planning and learning. Technical report, School of Computer Science, Carnegie Mellon University.

Cashmore, M., and Fox, M. 2010. Planning as qbf. *International Conference on Automated Planning and Scheduling Doctoral Consortium (ICAPS 2010)*.

Dershowitz, N.; Hanna, Z.; and Katz, J. 2005. Bounded model checking with QBF. In *SAT*, 408–414.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4):189–208.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proc. IJCAI-99*, 956–961.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *J. Artif. Int. Res.* 14:253–302.

Jussila, T., and Biere, A. 2007. Compressing BMC encodings with QBF. *Electr. Notes Theor. Comput. Sci.* 174(3):45–56.

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

Kautz, H.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proc. KR-96*, 374–384.

Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition*.

Mangassarian, H.; Veneris, A. G.; and Benedetti, M. 2010. Robust QBF encodings for sequential circuits with applications to verification, debug, and test. *IEEE Trans. Computers* 59(7):981–994.

Nguyen, X., and Kambhampati, S. 2001. Reviving partial order planning.

Penberthy, J. S., and Weld, D. S. 1992. Ucpop: A sound, complete, partial order planner for adl. 103–114. Morgan Kaufmann.

Peschiera, C.; Pulina, L.; Tacchella, A.; Bubeck, U.; Kullmann, O.; and Lynce, I. 2010. The seventh QBF solvers evaluation (QBFEVAL'10). In *Proc. SAT*.

Ridder, B., and Fox, M. 2011. Performing a lifted reachability analysis as a first step towards lifted partial ordered planning. In *In Proceedings of UK PLANSIG11 Workshop*.

Rintanen, J. 2001. Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *Proc. LPAR*, volume 2250 of *LNCS*, 362–376.

Rintanen, J. 2003. Symmetry reduction for sat representations of transition systems. In *Proc. AAAI*.

Rintanen, J. 2010. Heuristics for planning with SAT. In *Proceedings of the 16th international conference on Principles and practice of constraint programming*, CP'10, 414–428. Berlin, Heidelberg: Springer-Verlag.

Savitch, W. J. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.* 4(2):177–192.

Stockmeyer, L. J., and Meyer, A. R. 1973. Word problems requiring exponential time: Preliminary report. In *STOC*, 1–9.

Tate, A. 1976. Project planning using a hierarchic nonlinear planner. Technical report, Department of Artificial Intelligence, University of Edinburgh.

Towards Planning With Very Expressive Languages via Problem Decomposition Into Multiple CSPs

Uwe Köckemann and **Federico Pecora** and **Lars Karlsson**
Center for Applied Autonomous Sensor Systems (AASS)
Cognitive Robotic Systems Lab
Örebro University
{uwe.kockemann, federico.pecora, lars.karlsson}@oru.se

Abstract

The main contribution of this paper is a planning language that can handle temporal constraints, resources and background knowledge. We provide a solver for this language based on problem decomposition that uses constraint satisfaction problems (CSPs) as a common ground. We argue that the usage of more expressive languages not only allows a more direct modeling of planning domains, but can speed up the planning process as well. We also present an experiment in support of that argument.

Introduction

In this paper we present a novel way to model planning problems in a very expressive language that supports temporal constraints, reusable resources and explicit background knowledge. The language allows to model very fine-grained features of the domain. Its high expressiveness increases our ability to partition the various dimensions of the problem into sub-problems that can be solved by dedicated, state-of-the-art algorihtms. As an example, modeling the limited space of a location is easily done with resources, while it is cumbersome and inefficient using first-order literals; quantified temporal constraints can be used to easily compute the temporal placement of predicates in partial plans, including goal release times and potentially complex networks of predicates defining what happens during the execution of operators; and explicit background knowledge allows us to consider features like room connectivity separately from causal knowledge, thus reducing the burden of causal reasoning.

Problem solving in the proposed language is handled by decomposing problems into different types of *Constraint Satisfaction Problems (CSPs)* (Dechter 2003) that exchange information by pruning search spaces, adding constraints or reducing domains of one another. Each of the involved CSPs has a very specific reasoning purpose such as causal, operator applicability, temporal, resources or logic programming (LP). This leads to a decomposition where the individual solvers only see those parts of the problem that fit their purpose. Also, partitioning directly entails the ability to spread the computational load of the overall problem onto different sub-problem solvers, depending which metaphor is used to model particular domain features – e.g., modeling the occupancy of a block with a predicate (e.g., $\neg free(A)$) delegates the resolution of conflicts on block usage to a propositional planning solver, whereas modeling occupancy as resource usage allows to delegate this sub-problem to a scheduler.

We claim that our approach has two advantages, namely expressiveness in domain and problem modeling, and flexibility *wrt* how sub-problems are dedicated to different solvers. Regarding the latter advantage, we show specifically how modeling as resources elements of domains that are typically modeled as predicates in classical planning can actually provide a computational advantage. There are two reasons for this: first, an "implicit" resource model requires more predicates to model what happens during operator execution and second, it produces more conflicts over which a classical planning algorithm has to backtrack. In practice, we find that the time it takes for a scheduler to solve these conflicts is much smaller than coping with larger and more constrained planning problems.

Our approach does not make specific assumptions on the type of sub-problem solvers. In fact, any STRIPS planner could be used to solve the causal sub-problem of the overall planning problem. We argue, however, that constraint-based planners are particularly well suited because of the aforementioned possibility to exchange information by adding constraints or changing the domains of variables.

To show the advantages and a possible application of our system we provide an illustrative example of a domain and problem definition, and a solution. the next section discusses related research. We then introduce the domain and problem description language. The different CSP solvers used in our approach are introduced in the following sections, and their interactions are detailed. Finally, we will present some first results in the domain of the illustrative example which show the distribution of work between the different CSPs. We also present preliminary results on the comparison of two approaches to model the same domain, where one uses resources and the other one state-variables.

Related Research

The problem and domain description that is introduced here is in many respects similar to the Planning Domain Definition Language (PDDL) (Fox and Long 2003). The language described in this paper has higher expressivity and provides a straightforward way to model reusable resource usage and complex temporal constraints. Also, compared to tra-

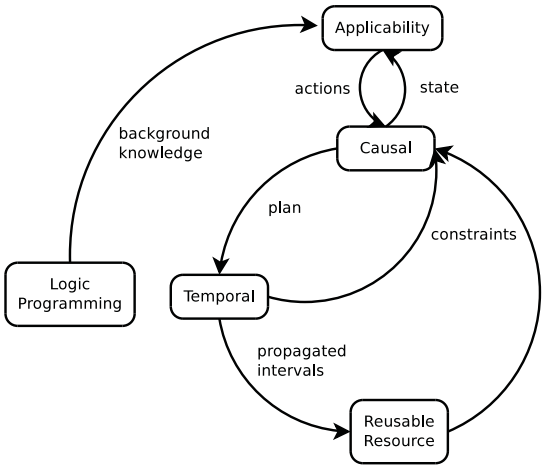


Figure 1: Interaction between different CSPs.

ditional planning languages, the proposed language allows a to model the "inside workings" of an operator. The language we propose is similar to the one described in (Ghallab, Nau, and Traverso 2004, Ch. 14), and is strictly more expressive as it supports resources and has explicit background knowledge. Furthermore, we allow a more convenient temporal modeling with quantified Allen's intervals (Allen 1984). The domain definition language is also similar to (Fratini, Pecora, and Cesta 2008), but extended with a notion of causality. In our approach we have chosen to employ a Graph Plan-like algorithm for causal reasoning.

Other approaches have explored its use for temporal planning, e.g., (Smith 1999; Long and Fox 2003) use the planning graph and durative actions. These languages are, however, less expressive than the one presented here, as our approach can model not only durative actions but also constraints between preconditions and effects. Furthermore, we allow the use of background knowledge to influence not only operator applicability, but temporal constraints and resource constraints as well. The approach in (Tsamardinos, Vidal, and Pollack 2003) extends Simple Temporal Problems (STP) (Dechter 2003) to Conditional Temporal Problems to cope with uncertainty. While the underlying temporal reasoning is similar, their approach goes into a different direction of conditional plans. The temporal POCL-based (partial order causal link) planner CPT (Vidal and Geffner 2004; 2006) uses constraints for optimal makespans. Our approach, on the other hand, is not concerned with optimality but rather with expressiveness. In that regard, it is similar to IxTeT (Laborie and Ghallab 1995), which is POCL-based instead of leveraging a planning graph and does not support explicit background knowledge as our language does. A recent approach combining Graph Plan with temporal reasoning is presented as TLP-GP by (Maris and Regnier 2008), which omit Graph Plan's mutual exclusion relations and let conflicts be sorted out by solving a disjunctive temporal problem. While being of similar temporal expressiveness, their approach lacks support of resources and background knowledge. Research on Graph Plan and resource scheduling was presented in (Srivastava and Kambhampati 1999). Their approach, however, does not cover temporal constraints and background knowledge and does not form a loop between the causal and the resource solver.

What distinguishes our approach from all previous similar approaches is the fact that we are not committing to any specific way of solving the different sub-problems. Neither do we commit to any way of combining their solutions. For these reasons our approach is open to be extended with further types of reasoning.

Domain and Problem Description Language

The proposed planning language uses a notion of preconditions and effects that does not entail any temporal information. Temporal information is added explicitly by means of temporal constraints in the form of quantified Allen's interval relations[1] that are linked to statements in temporal databases. Furthermore, we allow the use of reusable resources in addition to state-variables. A third notable fact is the use of background knowledge, that is not subject to temporal constraints. The notation is kept close to the one used in (Ghallab, Nau, and Traverso 2004, Ch. 14).

A temporal database

$$\Phi = (\mathcal{F}, \mathcal{T})$$

consists of a set of statements $\mathcal{F}$ and a set of temporal constraints $\mathcal{T}$ that establish a temporal context for these statements.

The basic building blocks of our language are *Statements*

$$s = (k, x, v),$$

where k is a unique key term identifying a temporal interval during which the statement holds. x is a first-order literal identifying either a state-variable or a reusable resource and v is the assigned value term. As an example take the following state-variable and resource assignments

$$\begin{aligned} s_1 &= (pos_0, location(rob_1), loc_1) \\ s_2 &= (space_0, space(loc_1), 2) \end{aligned}$$

stating that the interval pos_0 assigns value loc_1 to variable $location(rob_1)$ and interval $space_0$ assigns a resource usage of 2 to variable $space(loc_1)$.

The key k identifies a single temporal interval during which this statement holds. This interval is influenced by the temporal constraints $\mathcal{T}$ in Φ. If an interval becomes empty due to temporal constraints, we say that Φ is *temporally inconsistent*. A simple example of this kind of inconsistency with two statements and two constraints would be: a *before* b *before* a.

Each reusable resource has a maximum capacity. A resource conflict in a temporal database occurs when a resource is possibly used above its capacity during any period of time. Resource conflicts can be resolved by adding temporal constraints to Φ that enforce a usage smaller or

[1] For convenience, we added disjunctions of conceptually neighboring constraints such as "during-or-equals", as discussed in (Freksa 1992).

equal to the resources capacity. We say that Φ is *resource inconsistent*, if there is no temporally consistent database that resolves all resource conflicts. Reusable resources become available again after the interval of a statement that assigned a usage ends. Examples of this kind of resource include room, space or machine usage. A similar scheduling problem can be defined for values of state-variables. Here a conflict would arise if a state-variable can have more than one value in a given interval of time. If there are state-variable conflicts that can not be resolved we say there is a *state-variable inconsistency*.

A statement is ground when k, x and v are ground. A substitution $\sigma = \{x_1/v_1, \ldots, x_n/v_n\}$ on a statement s substitutes every part of the statement $\sigma(s) = (\sigma(k), \sigma(x), \sigma(v))$. It is important to substitute keys as well as variables and values, since there may be more than one statement with the same variable and value, but for different intervals.

Two statements s_1 and s_2 are equal iff $k_1 = k_2$, $x_1 = x_2$ and $v_1 = v_2$ and they are unifiable iff there is a substitution σ such that $\sigma(s_1) = \sigma(s_2)$. A statement s is supported by a temporal database Φ iff $\exists\sigma$ s.t. $\sigma(s) \in \mathcal{F}_\Phi$. Similarly, a set of statements $\mathcal{P}$ is supported by Φ iff $\exists\sigma$ s.t. $\sigma(\mathcal{P}) \subseteq \mathcal{F}_\Phi$. Finally, let $\theta(\mathcal{P}/\mathcal{F}_\Phi) = \{\sigma_i | \sigma_i(\mathcal{P}) \subseteq \mathcal{F}_\Phi\}$ be the set of all substitutions that match $\mathcal{P}$ to a subset of $\mathcal{F}_\Phi$.

A temporal context for/between statements is established by unary/binary *temporal constraints* that could impose, for example, flexible durations, release times or precedence constraints. A binary temporal constraint is defined as

$$c = (k_1, R, k_2, \mathcal{I})$$

and a unary temporal constraint as

$$c = (k_1, R, \mathcal{I})$$

where k_1 and k_2 are keys of statements and $R \in \{before, after, release, duration, \ldots\}$ is the type of relation. $\mathcal{I}$ is a sequence of flexible intervals that quantify the constraint. The following temporal constraints

$$\begin{aligned} c_1 &= (cook, before, dinner, [0, 20]) \\ c_2 &= (cook, duration, [30, 40]) \end{aligned}$$

state that the interval *cook* has to be between 0 and 20 time units before the interval *dinner*, and that the interval *cook* has a flexible duration between 30 and 40 time units.

A temporal database Φ_1 *supports* a second database Φ_2 if $\forall\sigma_i \in \theta(\mathcal{F}_{\Phi_2}/\mathcal{F}_{\Phi_1})$ their combination $\Phi' = (\mathcal{F}_{\Phi_1}, \mathcal{T}_{\Phi_1} \cup \sigma_i(\mathcal{T}_{\Phi_2}))$ is temporal, resource and state-variable consistent, meaning that all ways to support Φ_2 in Φ_1 are consistent with the temporal and resource constraints in Φ_1 without causing state-variable inconsistencies.

The *background knowledge* $\mathcal{B}$ consists of a set of first-order logic clauses modeling relations between objects that do not change over time. We use the notion $\mathcal{B} \models \mathcal{R}$ to state that background knowledge $\mathcal{B}$ entails a set of first-order literals $\mathcal{R}$.

An operator

$$o = (name, \mathcal{P}, \mathcal{E}, \mathcal{T}, \mathcal{R})$$

consists of a *name* statement, two sets of statements $\mathcal{P}$ and $\mathcal{E}$ for preconditions and effects, a set of temporal constraints $\mathcal{T}$ and a set of relational constraints $\mathcal{R}$. As an example see the *Move* operator in the next section, which presents an operator $Move(A, L1, L2)$ that uses state-variable at and resource $space$ to model an agent moving from one location to another temporarily occupying a corridor with a space requirement that depends on the agent's size. Note that all keys of statements are variables, so that unifying preconditions with a temporal database will identify which statements of Φ are considered preconditions for o.

An operator o is *applicable* to a temporal database Φ iff $\theta(\mathcal{P}_o/\mathcal{F}_\Phi) \neq \emptyset$ and its application $\Phi' = (\mathcal{F}_\Phi \cup \sigma_i(\mathcal{E}_o), \mathcal{T}_\Phi \cup \sigma_i(\mathcal{T}_o))$ for some $\sigma_i \in \theta(\mathcal{P}_o/\mathcal{F}_\Phi)$ is temporally, resource and state-variable consistent and $\mathcal{B} \models \sigma_i(\mathcal{R}_o)$.

Preconditions are not added to $\mathcal{F}_\Phi$ since, by definition of $\theta(\mathcal{P}_o/\mathcal{F}_\Phi)$ they are a subset of $\mathcal{F}_\Phi$ already. An operator is considered *ground* when $\mathcal{P}$, $\mathcal{E}$, $\mathcal{T}$ and $\mathcal{R}$ are ground. Ground operators are also called actions. Let $\theta(o/\mathcal{F}_\Phi|\mathcal{B}) = \{\sigma_i | \sigma_i \in \theta(\mathcal{P}_o/\mathcal{F}_\Phi) \wedge \mathcal{B} \models \mathcal{R}(\sigma_i(o)) \wedge \sigma_i(o)$ is ground $\wedge \forall_{e \in E_o} \sigma_i(e) \notin \mathcal{F}_\Phi\}$ be the set of all ground substitutions of operator o that are supported by Φ with respect to background knowledge $\mathcal{B}$. This set yields all ground substitutions for which an operator is applicable. The last part of the conditions states that the ground effect statements $\sigma_i(E_o)$ are not yet in Φ (assuring new unique keys for effects). Again, it is important to note that $\theta(o/\mathcal{F}_\Phi|\mathcal{B})$ will substitute precondition keys as well, creating one action for every combination of statements in $\mathcal{F}_\Phi$ that unify with $\mathcal{P}_o$. The notion of applicability can be extended to sets of actions, which might be necessary for cases in which no single action can be applied by itself, but a set of multiple actions becomes applicable. A planning problem

$$P = (\Phi_I, \mathcal{O}, \Phi_G, \mathcal{B})$$

is described using an initial temporal database Φ_I, a set of operators $\mathcal{O}$, a goal database Φ_G and the background knowledge base $\mathcal{B}$.

A plan

$$\pi = (\mathcal{A}, \mathcal{T})$$

is represented by a set of actions and temporal constraints that hold between them. These temporal constraints are established between the keys of the operator names. An application of π to a temporal database Φ yields a new temporal database

$$\Phi' = \gamma(\Phi, \pi) = (\mathcal{F}_\Phi \cup \bigcup_{a \in \mathcal{A}_\pi} \mathcal{E}_a, \mathcal{T}_\Phi \cup \mathcal{T}_\pi \cup \bigcup_{a \in \mathcal{A}_\pi} \mathcal{T}_a)$$

A plan π is supported by a temporal database Φ iff $\forall a_i \in \mathcal{A}: \ \mathcal{P}_{a_i} \subseteq \mathcal{F} \cup \bigcup_{j \neq i} E_{a_j}$ and $\Phi' = \gamma(\Phi, \pi)$ is temporally, resource and state-variable consistent. Note that the lack of order in which effects are added to the temporal database is intentional, since order is only imposed by temporal constraints (modeled in the operators or deduced to resolve resource contention.) For the same reason, the definition of support allows situations where two actions require each other's effects as preconditions. If for both actions there is a constraint stating that a precondition occurs before an effect, there will be a temporal inconsistency.

A plan π is a solution to the planning problem P iff Φ_I supports π and $\Phi' = \gamma(\Phi_I, \pi)$ supports Φ_G and $\mathcal{B} \models$

$\bigcup_{a \in A} R_a$. An example of a plan is presented in the next section. The representation is the same as for temporal databases, except for the assumption that statements in plans have to unify with names of operators.

It is obvious from these definitions that a direct approach to plan search is prohibitive, since only generating the set of applicable actions needs to find all possibilities to support an action's preconditions and each of these combinations has to be checked for temporal and resource consistency. This is why we propose an approach based on problem decomposition.

Before we describe the problem decomposition approach, we will provide an illustrative example that will also help as a running example in following sections and is used for the experiments later on.

An Illustrative Example

In this Section we will present an extended example of how a domain and a problem are defined in the language detailed in the previous section. The aim of this example is to show the capabilities of the suggested language and problem solving approach. Among these capabilities we demonstrate the possibility to use concurrency and to include resource contention to constrain the possible output schedules. We also show how background knowledge is used to model agents with different speeds and sizes. To keep the example short and to the point, we omit type and variable definitions. They are essentially the same as in PDDL (Fox and Long 2003). The only fact that should be mentioned is the capacity of the *space* resources, which is 1 for locations $space(loc1), space(loc2)$ and 2 for the corridor $space(cor1)$. By convention, constant terms are lower-case and variables are upper-case. All statements $s = (k, x, v)$ are defined similarly to how $P1$ is defined in *Move* (see below):

$$k :< x, v >$$

where $k := P1$, $x := at(A)$ and $v := L1$.

Domain

In our domain we have sets of agents who should transport objects between different locations. Locations are connected by corridors. Both locations and corridors are of limited capacity, allowing only a certain number of agents depending on their sizes. We use three operators: *Move*, *Pick* and *Put*. The *Move* operator moves an agent A from location $L1$ to $L2$.

```
OPERATOR Move(A,L1,L2)
  PRECONDITIONS:
    P1: < at(A), L1 >
  EFFECTS:
    E1: < at(A),C >
    E2: < at(A), L2 >
    E3: < space(C),  S >
    E4: < space(L2), S >
  TEMPORAL CONSTRAINTS:
    THIS DURATION [0,inf]
    E1 DURING THIS [0,inf] [0,inf]
    P1 OVERLAPS THIS [0,10]
    THIS OVERLAPS E2 [0,10]
    E1 DURATION [Tmin,Tmax]
    P1 MEETS E1
    E1 MEETS E2
    E1 EQUALS E3
    E2 EQUALS E4
    E2 DURATION [30,inf]
  RELATIONAL CONSTRAINTS:
    distance(L1,L2,D)
    speed(A,V)
    div(D,V,Tmin)
    add(Tmin,50,Tmax)
    size(A,S)
    corridor(L1,L2,C)
```

The *Move* operator has only one precondition prescribing that the agent A be at its starting location $L1$. As an effect of this operator, the agent changes its location and uses the resources $space(L2)$ and $space(C)$ with amount S. Next, the temporal constraints relate preconditions and effects in time. The special key *THIS* relates to the operator itself. We state that both $P1$ overlaps with the operator and that $E3$ is overlapped by the operator. $E2$ is during operator execution. Finally, the two *meets* constraints establish an order between preconditions and effects. Now, relational constraints are used to assure adjacency of the locations and to query the minimum and maximum ($Tmin$ and $Tmax$) amount of time the agent will occupy the corridor. Observe that the background knowledge also encodes knowledge about the time it takes each agent to execute an action. The amount of the space resource S is determined to be equal to the size of agent A.

The *Pick* and *Put* operators (below) require the agent to be at the same location as the object that is to be picked up/put down. In case of pick, we need to have the object at the right location ($P2$) and the agent should not hold any object ($P3$). After execution, the location of the object will be the agent's hand ($E1, E2$). For *Put* it is the opposite case, where the agent holds the object ($P2, P3$) as a precondition and as effects the object will be at the target location and the agents hand will be empty ($E1, E2$). For both operators the *speed(A,V)* predicate is used to calculate the execution interval for the action. Note that in case of *Pick* and *Put* we made use of *group keys*, which allow us to shorthand multiple temporal constraints. In this case *P2* and *P3* belong to group *P* and *E2* and *E3* belong to group *E*. Any temporal constraint using a group key will result in a corresponding temporal constraint for each group member. We omitted group keys in the formal model, since they do not add expressiveness and would have made the definitions in the previous section somewhat cumbersome.

```
OPERATOR Pick(A,O,L)
  PRECONDITIONS:
    P1:   < at(A), L >
    P2/P: < location(T), L >
    P3/P: < holds(A), nothing >
  EFFECTS:
    E1/E: < location(O), hand(A) >
    E2/E: < holds(A), O >
  TEMPORAL CONSTRAINTS:
    THIS DURATION [Tmin,Tmax]
```

```
    P OVERLAPS THIS [10,inf]
    THIS OVERLAPS E [10,inf]
    THIS DURING P1 [0,inf] [0,inf]
    P2 MEETS E1
    P3 MEETS E2
  RELATIONAL CONSTRAINTS:
    movable(O)
    speed(A,V)
    div(5,V,Tmin)
    add(50,Tmin,Tmax)
OPERATOR Put(A,O,L)
  PRECONDITIONS:
    P1: < at(A), L >
    P2/P: < location(O), hand(A) >
    P3/P: < hand(A), O >
  EFFECTS:
    E1/E: < location(O), L >
    E2/E: < hand(A), nothing >
  TEMPORAL CONSTRAINTS:
    THIS DURATION [Tmin,Tmax]
    P OVERLAPS THIS [10,inf]
    THIS OVERLAPS E [10,inf]
    THIS DURING P1 [0,inf] [0,inf]
    P2 MEETS E1
    P3 MEETS E2
  RELATIONAL CONSTRAINTS:
    movable(O)
    speed(A,V)
    div(5,V,Tmin)
    add(50,Tmin,Tmax)
```

Problem

Below we show an example problem. Type definitions are omitted. The background knowledge is formulated in Prolog and states that $loc1$ and $loc2$ are connected via corridor $cor1$ and have a distance of 4. Agents $rob1$ and $rob2$ have a size of 1 and a speed of 1 and 2 which influences the time it takes to execute movements. Furthermore, the *movable* predicate is used to state that objects a and b can be moved. Prolog was chosen because it is well suited for the relational knowledge that we would like to express in the background knowledge.

The last two lines connect the predicates *add* and *div* to the corresponding Prolog code. This is necessary because the relational constraints in the operators do not support Prolog syntax. Note that background knowledge can be provided both as part of the domain or of the problem. *add* and *div* would be part of the domain definition and the test (e.g. *corridor*) would belong to the problem.

As initial state we provide some statements about the whereabouts of agents and objects. Note that in the key $/s0$ we omitted the unique key, since we only care about the group key $s0$. A unique key is still generated for internal use. Statements $s2$ and $s4$ claim resources for the time (*equals* constraint) that the agents spend at their initial locations. $s5$ provides knowledge about a resource usage for $space(cor1)$ blocking the corridor for a certain time by assigning its capacity 2. Any plan that is to solve our problem has to respect this resource usage.

Finally, a simple goal is defined stating that objects a and b should be at $loc2$ and $loc1$ respectively. There is also a temporal constraints enforcing that the intervals of both these goals must be contain the interval $g1$ that starts between 200 and 210 and will last at least 20 time units. The statement $g1$ is used to synchronize all statements in group $g0$. Note that the given goal would not be solvable if the resource usage had been modeled with state-variables or relations, since this would have eliminated the possibility to execute the two location switching move operators in parallel because their preconditions would never have been satisfied.

```
BACKGROUND KNOWLEDGE:
  corridor(loc1,loc2,cor1).
  corridor(loc2,loc1,cor1).
  distance(loc1,loc2,4).
  distance(loc2,loc1,4).
  size(rob1,1).
  size(rob1,2).
  speed(rob1,1).
  speed(rob2,2).
  movable(a).
  movable(b).
  add(A,B,C) :- C is A + B.
  div(A,B,C) :- C is A / B.
INITIAL STATE:
  STATEMENTS:
    /s0:   < location(a), loc1 >
    /s0:   < location(b), loc2 >
    /s0:   < holds(rob1), nothing >
    /s0:   < holds(rob2), nothing >
    s1/s0: < at(rob1), loc1 >
    s3/s0: < at(rob2), loc2 >
    s2:    < space(loc1), 1 >
    s4:    < space(loc2), 1 >
    s5:    < space(cor1), 2 >
  TEMPORAL CONSTRAINTS:
    s0 RELEASE [0,0]
    s0 DURATION [0,inf]
    s1 EQUALS s2
    s3 EQUALS s4
    s5 RELEASE [50,50]
    s5 DURATION [50,50]
GOAL:
  STATEMENTS:
    g1:  < release() >
    /g0: < location(a), loc2 >
    /g0: < location(b), loc1 >
  TEMPORAL CONSTRAINTS:
    g0 CONTAINS g1 [0,inf] [0,inf]
    g1 RELEASE [200,210]
    g1 DURATION [20,inf]
```

Solution

Below we present a plan that solves the problem presented in the previous section. It contains six actions and four temporal constraints that impose an order on these actions. Figure 2 shows a Gantt chart of the scheduled actions in the solution plan. The time-lines in this chart were created using the earliest possible start-times and the earliest possible end-times of all intervals in the temporal database to which we applied the plan. Note that both *Move* actions start after the pre-scheduled *space* resource usage (see problem definition).

```
PLAN:
  ACTIONS:
```

```
  a1:  < Pick(rob1,a,loc1) >
  a2:  < Pick(rob2,b,loc2) >
  a3:  < Move(rob1,loc1,loc2) >
  a4:  < Move(rob2,loc2,loc1) >
  a5:  < Put(rob1,a,loc2) >
  a6:  < Put(rob2,b,loc1) >
TEMPORAL CONSTRAINTS:
  a1 BEFORE a3
  a3 BEFORE a5
  a2 BEFORE a4
  a4 BEFORE a6
```

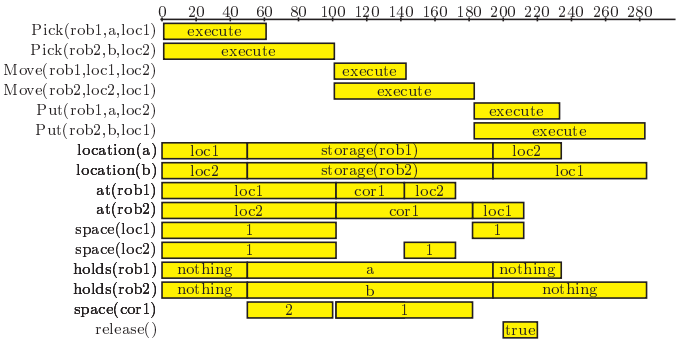


Figure 2: Gantt chart of the scheduled actions.

Problem Decomposition

We decompose the overall planning problem in the following way (see Figure 1). First, we solve the logic LP sub-problem which provides constraints for creating the set of applicable actions. Then the causal sub-problem is generated and solved for a (partial) plan. For each (partial) plan that is created based on this step we have the possibility to check if there are temporal or resource conflicts that can not be resolved. In case of such conflicts the according nodes and their successors are removed from the solution search. We will now discuss each part of this procedure in turn.

Causal CSP

The causal CSP is a CSP implementation of *Graph Plan* (Blum and Furst 1995) which ignores all temporal and resources constraints. The base of Graph Plan is the planning graph, which is created by generating the set of actions that can be applied in the initial state (first layer) and from that the set of propositions that can be reached from these actions. Based on this new set of reachable propositions the process is repeated for the second layer. During creation a set of mutual exclusion relations between actions and propositions is maintained, that contains actions that can not be executed in parallel and propositions that can only be reached by mutually exclusive actions. The creation of the planning graph is stopped when the set of propositions that was generated last contains all goal propositions and none of them are mutually exclusive (mutex) with each other. It also stops as soon as the latest created layer in the planning graph is equal to its predecessor. In this case, the planning graph has reached a fixed point. If a fixed point is reached, but the set of goals remains mutex or does not appear at all, the planning procedure returns failure without any search.

Once the planning graph is generated and given that no failure occurred during this step, we want to try and extract a solution from it. This solution extraction is done in a goal-directed manner from the last layer to the first. There are multiple ways to formulate solution search in the planning graph as CSPs (Kautz and Selman 1999; Kambhampati 2000; Lopez and Bacchus 2003). The one employed here is to solve the planning graph on a layer-by-layer basis. Starting with the last layer, we formulate a CSP where open goals are variables. Actions that achieve these goals are their domains, and mutual exclusion relations are constraints. In this way, a CSP solver will try to find an action for each open goal so that none of the mutual exclusion relations is violated.

Based on the solution to a layer n, a new CSP for layer $n-1$ is created, by using the preconditions of actions chosen in layer n as open goals for CSP $n-1$. In this way the following CSP attempts to enable the execution of actions on the next layer. The causal problem is solved if we reach the first layer and the remaining open goals are part of the initial state. If an inconsistent CSP is encountered we need to backtrack and choose a different solution to the previous layer's CSP. To allow backtracking, we maintain a search queue of CSP solutions to each layer. Every search step in solution extraction greedily chooses the search node that is closest to the initial state and generates new nodes for all solutions to the next CSP. This search queue is also the point where pruning can be performed, as will be described later. This way of extracting solutions from the planning graph is described in (Ghallab, Nau, and Traverso 2004)[p.128]. We will now describe how we extract a propositional planning problem from our overall problem for resolution with the procedure described above.

Extracting Graph Plan Operators

To extract a Graph Plan operators we convert all preconditions and effects to first-order literals (as detailed in (Ghallab, Nau, and Traverso 2004, Ch. 2)), while ignoring all temporal constraints and resource usages. In case of the *Move* operator, for instance, we would only use the two literals in statements *P1* and *E1* and ignore the resource assignments *E2* and *E3*.

To get the initial state, we query our temporal constraint solver for a certain time t_0 at which we wish to start executing the plan. For the goal, we just convert the according statements as was done for preconditions and effects of the operators. The background knowledge can be safely ignored at this stage, as is explained in the following section.

Background Knowledge & Applicability CSP

To be able to reason about background knowledge without burdening the causal CSP we decouple background knowledge from causal knowledge. This decoupling is realized by formulating the problem of finding the set of applicable operator instances as a CSP that uses constraints from the causal CSP (i.e., the state for which applicable actions

are searched) and from the background knowledge. Variables in this CSP are the open variables of the operator. Constraints are posed by preconditions combined with states and relational constraints combined with their valid ground instances (provided by LP). In CSP terminology, solutions to the applicability CSP define the domains of the variables of the causal CSP, which are sets of actions that achieve a certain open goal (i.e., a variable in the causal CSP).

Background knowledge is formulated in Prolog and we create and solve a Prolog query for each operator to generate the ground literals that can be used to apply the operator. This part is also a CSP where we solve for an operator's open variables given constraints posed in the background knowledge. If, for instance, in our *Move* operator we have $Move(rob1, loc1, loc2)$ and a constraint $size(rob1, S)$ the only assignment of S consistent with the background knowledge in the example is 1.

Pruning Based on Temporal and Resource Conflicts

There are two different ways of pruning when facing inconsistent plans. The first one is to remove every plan created based on the inconsistent plan from the solution extraction search queue. The second one is to add constraints to the planning graph to avoid that the same inconsistency occurs again. To accomplish this, it is possible to check pairwise actions on the same layer for conflicts or check if two consecutive actions cause a conflict. This could lead to the discovery of constraints such as the mutex

$$Move(rob1, loc1, loc2) \leftrightarrow Move(rob2, loc1, loc2)$$

if there were no way for these two *Move* actions to be executed in parallel without causing temporal or resource inconsistency. After adding new constraints in this way, all plans violating the new constraints can be pruned from the search queue. Another advantage is that the same inconsistency will not occur again, since the causal CSP was altered with knowledge from the temporal and resource CSPs.

Temporal CSP

The causal CSP can provide either a partial or a complete plan. Any plan that does not solve all layers of the planning graph is considered a partial plan. With respect to our language, the plans that are output by the causal CSP are not ground, as they do not instantiate the keys of statements in operator preconditions. There is, however, enough information in the order of actions (sorted by layers in the planning graph) to allow a unique mapping of precondition keys to either keys on the initial temporal database or effects that occurred on previous layers.

We start by querying the initial temporal database for all statements that hold at time t which gives us a subset C of the statements of Φ_I, that was also used as initial state for the causal CSP. If there are two statements s' and s'' with $x_{s'} = x_{s''}$ (i.e., assigning the same state-variable x) the one with the latest possible end-time is chosen. This way, the set C only contains one statement per state-variable.

Now we go through each layer of our plan (starting at the first one) and for each action create $S = \theta(\mathcal{P}_a/C)$ where by definition $|S| = 1$, since there is only one statement per state-variable and the only thing that remmains to be substituted is the key. This gives us a unique substitution $\sigma \in S$. Then we substitute the remaining keys of the effects in $\mathcal{E}_a$ with a new unique key and add them to σ as in $\sigma \cup \bigcup_{e \in E_a} \sigma_i$ s.t. $\sigma_i(e)$ is ground $\wedge$ $\sigma_i(e) \notin \mathcal{F}_{\Phi_I}$. Now C is updated as $(C - C_{old}) \cup E_a$ where $C_{old} = \{s | \exists e \in E_a \text{ s.t. } x_s = x_e\}$, so that for the next action $|S| = 1$ will still hold, but it will refer to the effects of the actions that were added before. The resulting ground operator is then applied to the temporal database.

The causal order between actions that is provided by the planning graph is thus translated to temporal constraints between preconditions and effects as specified is the operator definitions. By starting from a fixed initial state, we thus avoid the problem of deciding to which statement an operator's preconditions should be mapped. As an example of this, take the *Pick* operator which changes the $location(O)$ state-variable from $P2$ to $E2$. If our initial state (as in the example presented earlier) is $s0 : \langle location(a), loc1 \rangle$ and we add action $Pick(rob1, a, loc1)$ the key of precondition $P2$ is substituted with $s0$. The next operator that refers to $location(a)$ will in turn choose $E2$ as reference (more precisely, it will chose the new unique key that will be created to ground $E2$).

Finally, we add a goal action which has the goal conditions as preconditions and uses the temporal constraints of the goal. This allows to check the goal's temporal constraints in the same way the action's temporal constraints are checked. All temporal constraints can be translated to simple distance constraints between the start and end times of the involved statements. The result of this is a STP which can be solved in polynomial time by Floyd-Warshall's all-pairs-shortest-path algorithm (Dechter 2003)[Chapter 12].

Reusable Resource CSP

The reusable resource CSP is a precedence-constraint posting algorithm based on the Earliest Start Time Approach (ESTA) (Cesta, Oddi, and Smith 2002). The resource scheduling problem is cast, once again, as a specialized CSP in which variables represent sets of statements that over-consume resources, and their values represent temporal constraints which tease apart concurrent over-consuming statements so as to impose resource feasibility. More specifically, for a temporal database Φ

- Variables are sets of resource usage statements $S_r \subseteq \mathcal{F}_\Phi$ such that (1) all $(k_i, r, v_i) \in S$ overlap in time according to their earliest start-times, (2) they over-consume a resource r, i.e., $\sum_{(k_i,r,v_i) \in S} v_i > c_r$ where c_r is the capacity of resource r, and (3) all subsets of S_r do not over-consume the resource r.
- Values are temporal precendence constraints, (i.e., $before[0, \infty)$) to be imposed between a pair of statements in S_r so as to resolve the conflict arising from the concurrent execution of the statements in S_r.

Variables in the reusable resource CSP are called Minimal Critical Sets (MCS), as it is sufficient to impose a *before*

constraint between any pair of statements in S_r to regain resource feasibility. As in any typical CSP solver, the reusable resource scheduler performs a backtracking search over the variables which identify resource usage conflicts, choosing as values the possible alternative "resolvers" (precedence constraints) of these conflicts.

The reusable resource CSP avails itself of variable and value ordering heuristics for performance. While the details are outside the scope of this paper, it is worth mentioning that both these heuristics employ the notion of *temporal flexibility* (Cesta, Oddi, and Smith 2002), whereby variables which are most temporally constrained are chosen first, and resolving constraints (values) which least impact the temporal slack in the schedule are attempted first. It is also worth mentioning that in this work we employ a variant of the ESTA approach which performs a complete CSP search, thus guaranteeing the resolution of all conflicts.

The ESTA algorithm as briefly described above is employed both for resource conflict resolution and for enforcing state-variable consistency. This is achieved by simply changing the criteria used to identify a scheduling conflict (point (2) above), from sum of resource usage to different symbols.

Resulting Algorithm

Figure 3 shows the flow of the algorithms that combines all the parts described in the previous sections. Background knowledge and applicability are solved in the *init* step. After each step of planning, it can be decided if temporal and resource consistency should be checked for the current partial plan. If the current plan is a solution to the causal problem, we have to check temporal and resource constraints. Two possibilities we allow here are checking always and checking only for Graph Plan solutions. Ideally we would like to check only in case there is an inconsistency, since this is the only time we can prune the search space of solution extraction.

The proposed algorithm is sound but not complete *wrt* the language introduced in this paper. We argue that it is sound since the way a Graph Plan solution is used to map operators to statements together with the following temporal and resource constraints check fulfills the support criteria between Φ_I and π. Furthermore, the usage of the goal-action fulfills the support criteria between $\Phi' = \gamma(\Phi_i, \pi)$ and Φ_G. Finally, $\mathcal{B} \models \bigcup_{a \in A} R_a$ holds, since we used $\mathcal{B}$ to create constraints such that the applicability CSP only produces solutions that fulfill this requirement. In summary, we argue that the decomposition based algorithm is sound because:

1. a Graph Plan solution guarantees that the statements in all preconditions of the actions in the plan as well as the goal can be supported by some effect of some other action or the initial temporal database,
2. the procedure for putting the plan into the temporal CPS establishes a support for each precondition statement (including those of the goal action),
3. the temporal and resource CSPs then verifies that the resulting plan is temporal, resource and state-variable consistent.

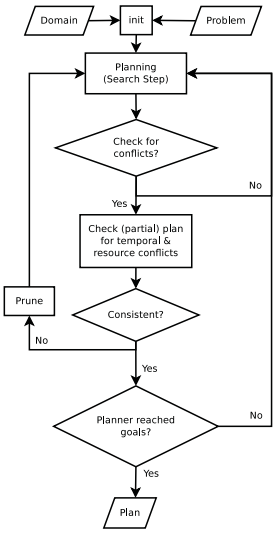


Figure 3: Flow chart of the decomposition-based algorithm.

Non-completeness can easily be shown by creating an example that requires concurrency that is not based on resources. We could for instance have a simplified *Move* operator that does not model corridors and, as a precondition, requires the target location to be free, which will be modelled with a boolean state-variable, rather than a resource. In this case, the *Move* operator would never be applicable in a problem with two locations, two agents and the goal to switch the agent's positions. Since a valid plan exists that uses two concurrent *Move* actions, but it cannot be found with our method, it is not complete.

It should be clear, however, that this is not the intended way to use the introduced language, since it was designed to model things "as they are" (i.e. not to use state-variables to model resources). Using resources to model resources makes the same problem solvable allows non-binary resources without any overhead and even finds solutions faster, as will be shown in the next section.

Experiments

In this section we present some preliminary results on problems from the domain that was described earlier in this paper. We randomly created planning problems for varying numbers of agents, locations and objects. For each configuration we created 20 problems and measured the mean time spent on the main parts of our decomposed problem: Graph Plan Expansion (GP-E), Graph Plan Solution Extraction (GP-S), Temporal CSP (T-CSP), Resource CSP (R-CSP) applicability CSP (APP-CSP) and LP. Note that APP-CSP is really part of GP-E, but we decided to measure them separately, to see how GP-E scales relative to APP-CSP.

In a first set of experiments, we check the runtime on a number of randomly generated problems for varying numbers of objects, agents, locations. We also investigate how much time is spent on each of the different solvers in the process. In a second experiment we compare two approaches to the same domain, where one approach uses resources to

model a bottleneck and the other one uses predicates.

When running our planner, we can choose to only run the temporal and resource CSPs after the causal CSP has found a solution or check these constraints for every single node in the Graph Plan search space. The second possibility becomes prohibitive for large planning graphs, but can lead to early termination in case of impossible goals. One example of such a goal would be a corridor that is smaller than the agent, but cannot be avoided in any solution plan. In the experiments presented here we only check consistency for causal solutions and leave a closer invesitigation of when/how often to check consistency as future work.

Analysis of Results

Most of the harder examples are dominated by Graph Plan solution extraction (GP-S), but the hardest set suffers from large temporal CSP (T-CSP) times, which is due to two particularly hard problems which contain a large number of valid Graph Plan solutions that are not resource consistent. These problems where of such a kind that even our pruning measures did not produces better results, since conflicts only appeared for solutions to the causal CSP.

The resource CSP (R-CSP) times are only large in the 3-3-3 set, where resource inconsistency also caused significant computation for the T-CSP. This is because a large number of resource-inconsistent plans where found before a feasible one. Here, it should be pointed out again that the R-CSP uses information from the T-CSP (temporal propagation to validate resolving temporal constraints), as was also indicated in Figure 1. On sets with no resource conflicts (with only one agent) the R-CSP time was almost zero, so it did not provide an overhead in cases where it was not needed.

The time it takes to query the background knowledge with LP is almost constant. This makes sense, since even for larger problems there is not too much addition to the background knowledge. Furthermore it does not contain clauses, except the ones for addition and multiplication, so there is very little inference at this stage. The time spent on the applicability CSP also grows moderately.

Resources *vs.* No-resources

As a second experiment, we compared the behavior of our planner solving the same domain modeled in different ways. More precisely, we would like to substantiate the claim that in some cases it is better to model resources as what they really are rather than to use state-variables or literals to simulate them. For this purpose, we used a domain derived from the one used in the previous experiment, but instead of randomly sized corridors and locations, all corridors are bottlenecks with unit capacity and all locations have unlimited space. Problems were again randomly created, but we made sure that they were the same for each of the three ways of modeling the domain, to avoid situations were one set suffers from a hard problem.

The first issue that came up when modeling without resources is that we had to split the *Move* operator into two operators, one moving into the corridor and one moving out of it. To capture the difference this makes we created the third experiment in which we use resources, but still split

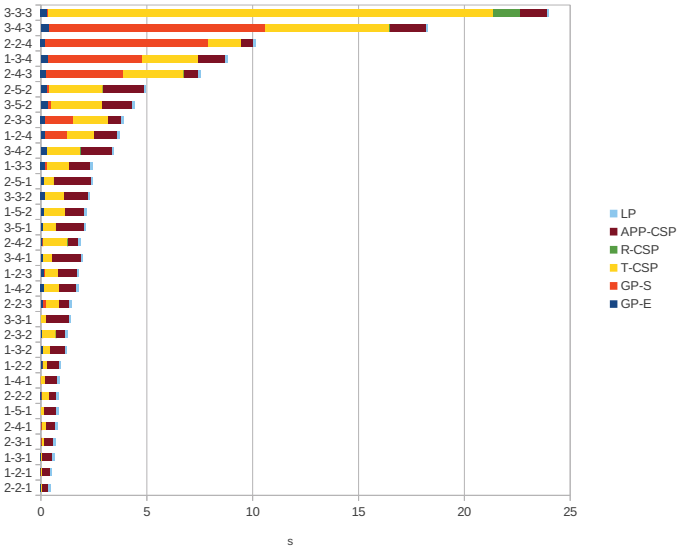


Figure 4: Results for random problems in the domain described earlier. The names of the experiments read $x-y-z$ for x agents, y locations and z objects.

the *Move* operator into two. In principle our language could model the problem with only one operator and no resources, but doing so yields a model that cannot be solved by the proposed method.

We compared the three models on two simple configurations (2 agents, 2/3 locations, 2 objects) and again over 20 random instances of this problem. For larger examples, the versions with two operators did not finish in a reasonable time. The reason for this is the fact that using two operators for *Move* leads to a deeper planning graph (more layers) and for solution extraction, we backtrack over solutions to each layer. In the version without resources, no two move actions were allowed on the same layer if they used the same corridor. So just for moving two agents we may need up to four planning graph layers. The two-operator version with resources was slightly less restricted, but essentially suffered from the same problem.

Figure 5 shows the results for the two configurations and three domain models. It can be seen that the approach using resources outperforms the others, which is mainly due to the smaller planning graphs. This example shows that resource scheduling can take away a huge burden from the causal problem by bearing the brunt of solving resource conflicts (for which it has powerful heuristics) that would otherwise over-constrain the planning graph.

Conclusion

The paper proposes a highly expressive planning language that can be used to model complex domains supporting background knowledge, temporal constraints and reusable resources. We showed how planning in this language can be done by decomposing problems into different types of CSPs. In so doing, we decouple background knowledge from causal knowledge and causal knowledge, in turn, from tem-

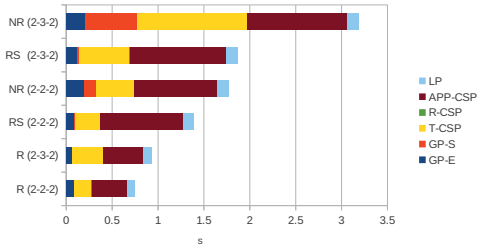


Figure 5: Comparison of the same domain modeled with three different approaches.

poral and resource knowledge. The individual sub-problem solvers communicate by exchanging constraints or providing part of each other's definition (e.g., domains of variables). We believe that a further exploration and generalization of the ideas applied here could be very useful. As evidence we provided a first set of experiments, demonstrating that it can be advantageous to choose a more expressive approach to domain modeling. Specifically, empirical evidence points to the fact that delegating the resolution of resource conflicts to a scheduler achieves much better run-times than encoding them as causal conflicts. We also provided a set of experiments to evaluate our method on varying problem sizes and investigated how much time was spent on each of the sub-problems.

In future work we would like to investigate the following directions:

- Further generalization of solving problems with different CSPs to allow adding other types of knowledge to be added in the same way.
- Adding goals to force the causal CSP to free resources. On the CSP level this would mean adding new variables.
- Experimental evaluation of the pruning / constraint checking trade-off. In this direction also more intelligent ways of deciding when to check temporal and resource constraints will be investigated.
- A SAT-based approach to search in the causal CSP would allow for better ways to control this part of the overall planning procedure. Arbitrary constraints can be added, as for example constraints spreading multiple layers.

Acknowledgement

This work was supported by the Swedish Research Council project "Human-aware task planning for mobile robots".

References

Allen, J. F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23:123–154.

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1):1636–1642.

Cesta, A.; Oddi, A.; and Smith, S. F. 2002. A constraint-based method for project scheduling with time windows. *Journal of Heuristics* 8:109–136.

Dechter, R. 2003. *Constraint processing*. Elsevier Morgan Kaufmann.

Fox, M., and Long, D. 2003. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:2003.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Freksa, C. 1992. Temporal reasoning based on semi-intervals. *Artificial Intelligence* 54:199–227.

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

Kambhampati, S. 2000. Planning graph as a (dynamic) csp: exploiting ebl, ddb and other csp search techniques in graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, IJCAI '99, 318–325. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, 1643–1649. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Long, D., and Fox, M. 2003. Exploiting a graphplan framework in temporal planning. In *ICAPS'03*, 52–61.

Lopez, A., and Bacchus, F. 2003. Generalizing graphplan by formulating planning as a csp. In *In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003), 954960*, 954–960. Morgan Kaufmann Publishers.

Maris, F., and Regnier, P. 2008. Tlp-gp: New results on temporally-expressive planning benchmarks. In *Tools with Artificial Intelligence, 2008. ICTAI '08. 20th IEEE International Conference on*, volume 1, 507 –514.

Smith, D. E. 1999. Temporal planning with mutual exclusion reasoning. In *In Proceedings of IJCAI-99*, 326–337.

Srivastava, B., and Kambhampati, S. 1999. Scaling up planning by teasing out resource scheduling. In *Proceedings of the 5th European Conference on Planning: Recent Advances in AI Planning*, 172–186. London, UK: Springer-Verlag.

Tsamardinos, I.; Vidal, T.; and Pollack, M. E. 2003. CTP: A new constraint-based formalism for conditional, temporal planning. *Constraints* 8(4):365–388.

Vidal, V., and Geffner, H. 2004. Cpt: An optimal temporal pocl planner based on constraint programming. In *IPC (ICAPS) 2004*.

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

A Constraint-based Framework for AEOS Mission Planning and Scheduling

Zhenyu Lian, Yuejin Tan, Yingwu Chen, Feng Yao, Jufang Li
College of Information System and Management
National University of Defense Technology
Changsha, Hunan Province, 410073, P. R. China

Abstract

Most mission planning problems in space are integration of planning and scheduling problems, as is also the case for the mission planning problem of Agile Earth Observation Satellites (AEOS). An AEOS problem is a partial satisfaction planning problem, or an oversubscribed scheduling problem, with temporal dependence characteristic, logical constraints, resource constraints and domain specific constraints. Current planning algorithms and frameworks, such as CNT, CAIP and OMP, could not handle all the features of our problem, especially the temporal dependence characteristic. In this paper, a constraint-based framework CPSSA is introduced which addresses three aspects, i.e. concepts for problem description, a new constraint model translating the problem description into a constraint representation and a solution technique integrating heuristic search and constraint reasoning processes. In the problem description, we incorporate concepts of HTN planning and RCPSP scheduling into our framework to represent the mission planning problem of AEOS. In constraint model, we introduce a new activity graph, of which the variables and constraints bridge the gap of problem description and constraint representation for a more flexible solution technique. The activity graph of constraint model is also a link between heuristic search and constraint reasoning, which interleave with each other in the solving process. As an implementation of CPSSA, a Space Mission Planning and Scheduling engine (SMPS) suitable for the AEOS domain is developed.

Introduction

An important function issue for a space agency like the China Center for Resource Satellite Data and Applications (CRESDA) is the management of its earth observing satellites (EOS) (Wang and Reinelt 2010). Following the development of satellite techniques, a newer agile earth observing satellite (AEOS) (Lian et al. 2011) brings more imaging opportunities and higher resolution available to users, which will be applied to future environmental and disaster monitoring and prediction. However, current scheduling systems (Wang et al. 2011), just for EOS, cannot support this new type of satellite. Therefore, a new mission planning framework for AEOS, which is also compatible with EOS, is introduced in this paper.

Equipped with optical or radar instruments, with gyroscopic actuators with which the satellites are able to move freely around their inertial center along the three axes (yaw, pitch and roll angle) (Beaumet and Verfaillie 2007), AEOSs are placed on sun-synchronous, low altitude, circular orbits around the earth (Beaumet, Verfaillie, and Charmeau 2007). There are attitude control model, memory model, battery model and other models in each satellite.

The mission planning system of AEOS first receives user's requirements, and then builds regularly an activity plan for satellites and ground stations, which satisfies all the physical constraints and meets as many requirements as possible. Generally, users submit image requests of several targets, and then operators decompose the targets to strip areas as required, each of which can be imaged by satellites within one pass. Each strip is usually defined as a task with two activities, i.e. imaging and downloading. In reality, there would be tasks including more than two activities. For example, to get stereo images of a certain target there must be two or three imaging activities and a downloading activity. These tasks will not be involved in this paper. There are many opportunities for each strip to be imaged or downloaded in satellite's planning horizon. In each opportunity, there are three attributes, i.e. time window, resource requirements and preconditions. In AEOS domain, an imaging/downloading activity can be done by many satellites, and for each satellite there are many time windows since its periodic movement. The preconditions of activity execution, about state of resource or environment, would be different among time windows. If a task/strip is inserted into a plan, there must be an opportunity being selected for each activity of the task. Besides, there are also assistant activities, i.e. slewing, sun-pointing, earth-pointing, camera-on, camera-off and file-erasing activities. The assistant activities are not directly related with any task, but necessary for some activities of tasks and satellites' working. There are strict mutually exclusive and concurrency constraints in satellites and stations, e.g. slewing, imaging, sun-pointing and earth-pointing are mutually exclusive, imaging and downloading could be concurrent in real-time transferring mode and mutually exclusive in routine mode.

AEOS is different from EOS as follows. For AEOS, agility means significant flexibility in assigning start time of imaging/downloading within time windows (Lian et al. 2011), while start time is determinate for EOS. This is illustrated in Figure 1, which shows the observable scopes of non-agile and agile satellite at a time. For a non-agile satellite (see Figure 1 (a)), the observable scope is just a line along the roll axis, which means observation is only possible when the satellite flies over the target ground area. So the realization window of an observation is fixed. For an agile satellite (see Figure 1 (b)), the observable scope is an area along the roll and pitch axes, which means observation is possible before, during, or after AEOS flying over the target ground area. Thus, after selecting an opportunity for a activity, the domain of its start time variable in EOS has one value, while the domain in AEOS is an interval. More freedom in terms of observation is the main advantage of agility.

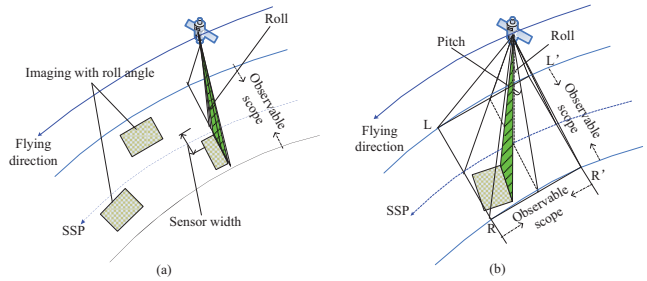


Figure 1: Observable scopes of non-agile and agile satellite

However, more challenges in terms of operation scheduling are also the main "disadvantage" of agility. The characteristics of an AEOS problem are listed as follows.

1. Partial satisfaction. The AEOS mission planning and scheduling problem is inherently an oversubscribed constraint optimization problem. Too much image requests need to be considered relative to the ability of satellites and ground stations. Thus, mission planning system should satisfy as many requests as possible.

2. Temporal dependence (Lian et al. 2011). There are two types of decision variables for a task in AEOS, i.e. an alternative variable (whether to schedule or not) and a time variable (when to start). Within the time window of an opportunity, there are potentially infinite numbers of solutions for AEOS to fulfill a given task, since time is continuous. Pointing attitude of imaging is a function of its start time. And there must be a slewing activity for different pointing attitudes of adjacent activities. The slewing time and power consumption depend on the attitude difference (Lian and Xing 2011). This complex dependence relation, defined as temporal dependence, could not be directly described by internal constraints, as in Figure 4. So we would define custom constraints, as in constraint model section later.

3. Logical constraint (Grasset-Bourdel 2010). There are some assistant activities added by preconditions of activity execution, which are not directly related with task. For example, a slewing, or a power production (sun-pointing (Lian and Xing 2011)), is inserted, if there is an attitude requirement, or a power requirement.

4. Special constraint (Wang and Reinelt 2010). There are some special constraints on the satellites of China. For example, cumulative imaging time within one circle cannot exceed 30 minutes, cumulative working time within one circle cannot exceed 40 minutes, the count of slewing cannot exceed 5 times, etc.

Integration of planning and scheduling in AEOS domain, is significantly necessary for its potential gain as in Figure 2. If there is no planning ability, we can find an 'optimal' solution for a simple EOS problem, but there would be losses of tasks for a complex AEOS problem with much more assistant activities. In Figure 2, the 'optimal' schedule just includes imaging1 and imaging2. If we could insert a power production activity, i.e. a sun-pointing activity, imaging3 could be executed. Therefore, we could accomplish much more tasks by integrating planning into scheduling.

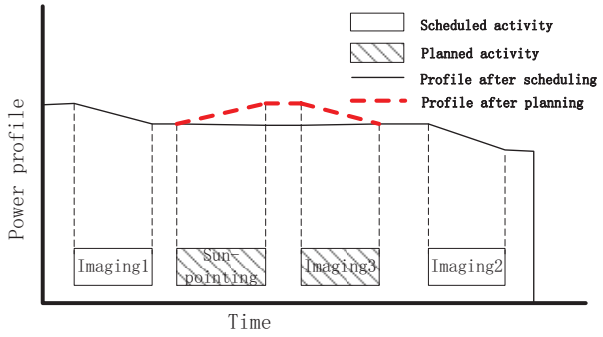


Figure 2: Potential gain of integrating planning into scheduling

In short, an EOS problem is just a combination optimization problem, while an AEOS problem is a problem hybrid of combination optimization, continues temporal reasoning and logical reasoning.

The early planning algorithms for AEOS (Lematre et al. 2002) problems plan imaging activities and downloading activities sequentially, and then check energy and memory constraints. They did not consider logical and special constraints of the AEOS problem. In case of constraint violation, some tasks are removed until constraint satisfaction, leading potentially to a sub-optimal solution (Grasset-Bourdel, Verfaillie, and Flipo 2011). Then, a chronological forward search algorithm (Grasset-Bourdel, Verfaillie, and Flipo 2011) with dedicated decision heuristics and constraint checking was developed, and it could produce a good solution in limited time. However, the algorithms are limited in optimization for its local search feature (limited look ahead and backtrack) and hard to extend to more general problems, especially our AEOS problem with temporal, logical and special constraints.

The early general frameworks for AEOS, e.g. CNT (Pralet and Verfaillie 2008), could only describe a simple AEOS problem as ground constraints and variables. So it is hard for CNT to handle real problems with much more physical constraints and tasks. CAIP (Frank and Jonsson 2003),

a more general framework used in practical scenarios, is built around temporal intervals and attributes with NDDL language. Another framework, OMP (Fratini, Pecora, and Cesta 2008) is based on state variables and resources with DDL.2 language. These architectures have always emphasized the use of a rich representation planning language to capture complex characteristics of domains (Fratini, Pecora, and Cesta 2008). The NDDL and DDL.2 derived from DDL (from HSTS (Muscettola 1994)) have been very fruitful, but it is painful to complex physical constraints (Frank et al. 2011). And CAIP's goal is to find a complete solution with an incomplete initial state, which is in nature a constraint satisfaction problem, not suitable for partial satisfaction problem. But our AEOS usually cannot satisfy all the task requirements. Besides, you must define complicated configuration rules or patterns to represent state variable's transition in CAIP or OMP. But state variable's transition is complicated and hard to describe our real problem. For example, sun-pointing activities, not related to any task, could supply power for satellites to execute more activities. So a sun-pointing activity will be inserted and adjusted to guarantee urgency tasks in sunniness periods of satellites, when power is not sufficient. Configuration rules or patterns cannot handle this conditional precondition with time and resource's state.

The scheduling engine OptTime (OptTime 2011) embedded in the STK Scheduler, is built around fixed tasks without planning ability. The OPL, in ILOG (ILog 2003) is built with ground constraint representation. Although it is very flexible for many domains, we must describe all the problem details. For our partial satisfaction problem, it is very time-consuming because it will try all tasks before providing a rational result.

However, the temporal dependence characteristic in the AEOS problem causes the interlacing of physical constraints and internal constraints, and cannot be resolved in the above algorithms and frameworks.

We continue the trend set with these earlier efforts. In this paper, Componential Planning and Scheduling for Space Architecture (CPSSA), a constraint-based framework that explicitly supports all the characteristics of our AEOS problem, is introduced. This framework unifies planning and scheduling problems with basic concepts, i.e. tasks, activities, opportunities, resources, preconditions, etc. In the problem description, unrelated tasks to be scheduled or not describes partial satisfaction of problems. Activities describe key operations for tasks' fulfillment. Opportunities identify logical requirement and temporal constraints of activities. And resources handle logical constraints and special constraints with custom defined constraints. The problems described with above concepts are translated into constraint representation, and solved by interleaving heuristic search and constraint reasoning. CPSSA is built as a reference for the development of planning and scheduling system for AEOS.

The paper is organized as follows. We first introduce problem description and a new constraint model of CPSSA. Next the solving component in CPSSA is introduced, and then a Space Mission Planning and Scheduling engine (SMPS), which is an implementation of CPSSA, is presented. Finally, we draw some conclusions about the proposed framework.

Problem Description

Basic Concepts

Task and Activity In classical planning, an action is an abstract operation with preconditions and effects, and a series of actions must be instantiated, forming the plan that transforms the initial facts into its goal. In classical scheduling, a job equals a sequence of activities, each of which is an operation with setup time, predecessors and successors. A set of activities belong to jobs must be allocated limited shared resources through time. In our framework, there are a set of tasks (similar with job and goal) according to requirements, each $task$ could be decomposed into a series of key activities, and each $activity$ is an operation with preconditions, predecessors and successors. Meanwhile, there are also assistant $activities$, which are abstract operations and could be instantiated to support tasks. Resources are too rare to meet all tasks, thus only part of $tasks$ (goals) could be achieved. In the final plan, for each key $activity$ of scheduled tasks, there must be a valid $opportunity$ (introduced in next paragraph) to execute, a rational start time in time windows, predecessors and successors satisfied by activities of the plan and preconditions satisfied by assistant activities, and for each resource, there is a rational sequence of activities within its capacity.

Opportunity Derived from RCPSP (Brucker et al. 1999), a new $opportunity$ concept replacing mode concept is introduced here. In Multi-Mode RCPSP (Brucker et al. 1999), each mode reflects a feasible way to combine a duration and resource requests that allow accomplishing the underlying activity. An activity's $opportunity$ has attributes of time window, resource requirements, preconditions (domain dependent), etc. It allows accomplishing the underlying activity (activities, for some opportunities being shared among activities, e.g. downloading opportunities). An activity must be performed in one of its opportunities. And an activity can be executed only if it is assigned to a valid opportunity with enough setup time and rational preconditions in the plan and satisfies predecessor and successor relations. Opportunity change and preemption are not permitted during activity execution.

Precondition Furthermore, we extend opportunity to handle causal constraint with preconditions, which are derived from preconditions of an action in HTN planning (Ghallab, Nau, and Traverso 2004). The $precondition$ implies how an activity executes in the current opportunity and which assistant activities should be executed before.

Thus, opportunity is defined for the activity to identify execution opportunity, which has attributes of feasible time window, resource requirements, preconditions, etc. In the space domain, the opportunity is relatively limited within a time horizon (e.g. daily).

Resource The concepts of resource are similar in both planning and scheduling definition. We distinguish

$reservoir$ resource, $reusable$ resource and $timeline$ resource, as in (Bedrax-Weiss, McGann, and Ramakrishnan 2003). In CAIP (Frank and Jonsson 2003), CNT (Pralet and Verfaillie 2008), OMP (Fratini, Pecora, and Cesta 2008), timeline means a component or thread of one distributed complex control system. At any time of one timeline, there would be no more than one activity to be executed, i.e. the capacity of Timeline is 1. In our framework, timeline is similar, which is repressed by a reusable resource with an availability of one unit per period. For reservoir resource like battery, capacity consumption will continue after activity execution. For reusable resource, capacity will only be used during activity execution. Each resource maintains its own activities and profile. Also, our resources are able to generate new assistant activities in order to satisfy preconditions of activities.

Plan or Schedule A $Plan$ (or synonymously, $Schedule$), is an activity graph made up of activities. Temporal relations and logical requirements for all activities in the plan are satisfied. Each key activity is assigned an opportunity, and physical constraints and preconditions of activities are satisfied on each resource.

Planning Problem

Assume that π is the final plan/schedule. A planning problem is given by a six-tuple $\Pi =< TK, A, V, O, R, C >$

TK is a set of tasks, $\{tk_1, tk_2, ..., tk_{tkn}\}$, where tkn is the number of tasks. Each $tk_i \in TK$ can be decomposed into a set of key activities A'_i, $\{a_{i,1}, a_{i,2}, ..., a_{i,ain}\}$, where ain is the number of activities in task tk_i. $A' \subseteq A$.

A is a set of activities. Some activities are assistant activities $A*$, where are independent of any task (produced by preconditions of A'). Each $a \in A$ is at least associated with three variables, start, end and duration. There may be some variables to satisfy constraints, e.g. slew angle. Activity a is applicable if and only if its preconditions satisfies $Pre(a) \subseteq \pi$, and $a_{i,j} \in \pi \Rightarrow ao_{i,j} = o_{i,j,k} \Leftrightarrow Start_{a_{i,j}} \in TW_{i,j,k} \cup a_{i,j} \in Res_{i,j,k} \cup Pre_{i,j,k} \in \pi \cup Eff_{i,j,k} \in \pi$.

V is a set of variables, each $v \in V$ is associated with a finite domain $dom(v)$ and also not all variables related with activities.

O is a set of activity opportunities, each $o \in O$ is associated with activities. Opportunity o is exclusively belong to one activity, e.g. imaging opportunity, or shared with several activities, e.g. downloading opportunity. $O_{i,j}$ is the set of opportunities for an activity $a_{i,j}$, $\{o_{i,j,1}, o_{i,j,2}, ..., o_{i,j,on}\}$, while on is the number of elements in the set. There are four attributes, time window $TW_{i,j,k}$, resource requirement $Res_{i,j,k,r}$ and quantity $Qua_{i,j,k,r}$, preconditions $Pre_{i,j,k}$ for each opportunity o.

R is a set of resources,, each $r \in R$ is associated with a fluent variable, and a conditional trigger to response to preconditions of activities. $\{r_1, r_2, ..., r_{rn}\}$, while rn is the number of elements in the set. $a_{i,j} \in r \Rightarrow profile_t \in [0, capacity_r], t \in [0, HT]$, HT is the end of Horizon. Resource r checks its state and activity's preconditions, and generates new activities, $Pre(a) \subseteq \pi \Rightarrow a' \subseteq \pi$

C is the constraints of variables in V.

Then a plan π can be defined as a triple-tuple, $\pi =< A, V, \preceq >$, A, V are the set of activities and variables in plan, respectively. $\preceq$ is the order relation of activities in plan. Any task is scheduled, every key activity of it must be scheduled, i.e. $\forall tk_i \in \pi \Rightarrow a_{i,j} \in \pi, \forall a_{i,j} \in tk_i$. Activity graph AG, as in Figure 5, means a directed graph $< A, E >$, A is the set of activities, E is the set order relations in A. AG is a representation of activity relations of π, so $a_{i,j} \in AG \Leftrightarrow a_{i,j} \in \pi$

And optimization was generally assumed to be unimportant in planning, but our problem is an partial satisfaction problem, and optimization criteria for Π is $h(\pi) = max \sum Pri_{tk_i}, tk_i \in \pi$.

In this way, problems that can be represented by the above concepts will be able to be solved in our framework.

Example

In the AEOS domain, we use basic concepts defined above to describe reality problems as in Figure 3. We introduce "Get Image" of a certain target or strip as a task, which can be decomposed into two key activities, imaging and downloading. There are also assistant activities, such as sun-pointing, slewing, etc., produced by resources. The key activities have their own opportunities, imaging opportunity and downloading opportunity. Imaging opportunity identifies time windows of satellites where the imaging of the target can take, and the amount of memory being used, attitude and power preconditions, etc. Downloading opportunity identifies which satellite can download images to which station, how many memories will be transferred, how many power will be consumed. For physical reasons, the imaging opportunity is exclusively belong to a imaging activity, while the downloading opportunity is shared with many downloading activities, i.e. images of imaging activities could be downloaded together in a downloading opportunity of a satellite with a ground station. In each opportunity, there are temporal attributes, e.g. time window, predecessor and successor, and logical attributes, e.g. power and attitude preconditions. The resources could consider above preconditions and generate new assistant activity instantiations into current plan based on its state profile, and be responsible for checking special constraints with third party code. An attitude precondition implies a slewing activity if attitudes of adjacent activities are different, which is determined by attitude control resource, while a power precondition is handled by battery resource. For example, one imaging activity has preconditions, i.e. a 12 kwhr power precondition, a (0, 2, 3) attitude precondition (expressed in terms of Euler angle $(Yaw, Roll, Pitch)$). To satisfy a power precondition, battery resource of the satellite will insert or not a power production activity (sun-pointing) based on its state. To satisfy an attitude precondition, attitude control resource of the satellite will insert or not a slewing activity based on attitude difference of prior activity and this imaging activity. Simply, activities claim preconditions of power or attitude, and then resources respond it with new activity based on self-state.

As in Figure 3, attitude, sensor, battery, memory and station are defined as 'Timeline', 'Reservoir' and 'Reusable' resources. A satellite is made up of attitude, sensor, battery, memory, etc. In reservoir and reusable resource definition,

the numbers, e.g. 100 and 30, are capacities of the resources. Slewing and sun-pointing is defined separately as abstract activities. There are three tasks, getImage1, getImage2 and getImage3. In task definition, the numbers, e.g. 32, 13, etc., are the coordinates of targets. GetImage1 includes two activities, i.e. imaging1 and downloading1, which are concrete activities and have their own opportunities. Downloading opportunities $do1, do2, do3$ are shared with three downloading activities. Successor attribute of activities identifies order relations of concrete activities. Opportunity attribute of activities is a list of pending opportunities, which can be shared with other activities. The resource requirement definition, i.e. Res, is about resources and their consumption quantities. The precondition definition, i.e. Pre, is about state of resources, e.g. (Attitude, (0, 2, 5)) identifies that attitude resource is requested in attitude (0, 2, 5) before the activity execution.

```
//Resource definition
Timeline attitude1, sensor1;
Reservoir battery1(100), memory1(30);
Reusable station1(2), station2(3);
satellite1 = {attitude1, sensor1, battery1, memory1};
//Task definition
Task getImage1(32,13), getImage2(12,34), getImage3(53,21);
//Activity definition
Assistant Activity slewing, sun-pointing;
getImage1.Activity imaging1, downloading1;
getImage2.Activity imaging21, imaging22, downloading2;
getImage3.Activity imaging3, downloading3;
imaging1.Successor.Add(downloading1);
imaging21.Successor.Add(downloading2);
imaging22.Successor.Add(downloading2);
imaging3.Successor.Add(downloading3);
//Opportunity definition
Imaging Opportunity io1, io2;
Downloading Opportunity do1, do2, do3;
imaging1.Opportunity.Add({io11,io12});
imaging2.Opportunity.Add({io21,io22});
imaging3.Opportunity.Add({io31,io32});
downloading1.Opportunity.Add({do1,do2,do3});
downloading2.Opportunity.Add({do1,do2,do3});
downloading3.Opportunity.Add({do1,do2,do3});
io1.TW = [0,3]; //time window
//resource requirement
io1.Res = {(sensor1,1),(power1,40),(memory1,30)};
io1.Pre = {(Attitude,(0,2,5)),(Power,80)};
io2.TW = [6,9];
io2.Res = {(sensor1,1),(power1,40),(memory1,30)};
io2.Pre = {(Attitude,(0,2,5)),(Power,80)};
//other imaging opportunities are curtailed.
do1.TW = [7,10];
do1.Res = {(power1,20),(station1,1)};
do1.Pre = {};
do2.TW = [12,17];
do2.Res = {(power1,20),(station1,1)};
do2.Pre = {};
do3.TW = [18,23];
do3.Res = {(power1,20),(station1,1)};
do3.Pre = {};
```

Figure 3: The problem description of an AEOS example

Constraint Model

The concepts defined above are not sufficient for our constraint-based framework, since the constraint and variable are also central concepts in ground level. This is done by mapping each activity graph (partial plan) to a constraint representation. The advantage of such a representation is that any algorithm which solves Dynamic Constraint Satisfaction Problem (DCSP) can be used to solve our integration problem. However, there are some new requirements in constraint management in our AEOS problem.

As in Figure 4, for each activity, there are three basic variables, 'start', 'end' and 'duration', and a constraint 'AddEq' (start + duration = end). We can define customized variables (e.g. angle) and constraints between these variables and other variables of a specific activity. Global constraint between variables of different activities can also be defined.

Following a task' schedule state and a activity's generation or adjustment, a constraint model maintenances their variables and constraints. For **partial satisfaction** feature of problems, the constraint model will add or remove variables and constraints of tasks and activities, and the domains of variables will be relaxed or restricted.

Attitude angle of an imaging activity is a function of its start time, $I2\_angle = Func(I2\_start), I1\_angle = Func(I1\_start)$. This function cannot be translated into constraint representation for its complex spatial geometry relations. So third part code must be used in this constraint model (Grasset-Bourdel, Verfaillie, and Flipo 2011). The S_dur, duration variable of slewing activity depends on $I1\_angle$ and $I2\_angle$. S_dur also has influence on $I1\_start$ and $I2\_start$. This is a **temporal dependence** problem, which cannot be described in early framework, e.g. CAIP and OMP. In our framework, the third part function is also derived from basic expression. Thus, physical constraints can also be embedded.

The slewing activity is also an example of handling **logical constraints** in planning. For instance, attitude precondition of $I2$ is $I2\_angle$, and the attitude resource will add a slewing activity by judging the condition, "$I2\_angle \neq I1\_angle \Rightarrow$ add a $slewing\ activity$". Similarly, assistant activities, like slewing and sun-pointing activities, can be heuristically added by checking the conditional constraints, which depend on time or activity's precondition.

The **special constraints** as in Figure 3, are handled by the derived class of resources. For example, the cumulative imaging time could be defined in the camera definition and checked in constraint reasoning. These special constraints are also derived from basic constraint, but a black-box constraint checking function.

Solving

Our AEOS problem is translated into constraint models, then can be solved with mature solution techniques of constraint programming, such as constraint propagation, consistent checking. As a partial satisfaction problem, we use mature heuristic search algorithms within high level concepts. Thus, the solving process is composed with two parts, i.e. heuristic search and constraint reasoning, and an activity

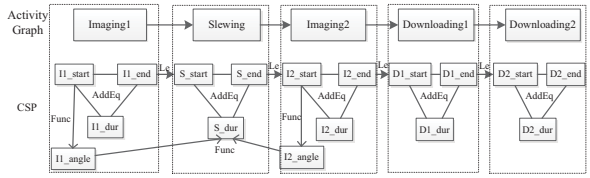


Figure 4: An activity graph and its constraint representation of a simple AEOS problem

graph is a bridge of them. In heuristic search, we construct and revise the activity graph with causal constraints. In constraint reasoning, we resolve temporal conflicts and resource flaws (Frank and Jonsson 2003). However, heuristic search and constraint reasoning are not sequential stages, but interleaved with each other.

Heuristic Search

Assuming that the initial condition is empty and the goal is to be accomplished as many tasks as possible. If a task comes into conflict with scheduled tasks, the task will be unscheduled. So the tasks are tried to be scheduled in a heuristic order. Optimization can be guaranteed by iterated search with different order. Key activities of each task are sequentially scheduled with heuristic information used to search a feasible opportunity, and then resources response to the opportunity's preconditions by generating or adjusting assistant activities. In this way, we can get an activity graph satisfying the logical constraints and precedence constraints, as in Figure 5. The heuristic search includes three parts, i.e. scheduling order of tasks, opportunities of activities and assistant activities for logical satisfaction of preconditions. Following the feedback from the temporal reasoning and resource reasoning in constraint reasoning (discussed in the next section), an iterative heuristic search discovers another activity graph with different tasks, alternative opportunities of key activities or different assistant activities generated.

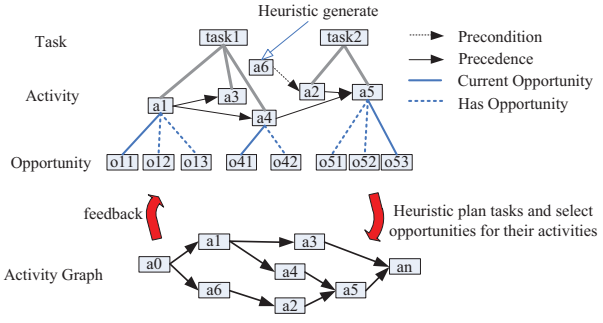


Figure 5: Heuristic search process

Constraint Reasoning

Given an activity graph from 'heuristic search' process, constraint reasoning handles the temporal and resource constraints in the activity graph, as in Figure 6. The temporal constraints are represented with a Simple Temporal Network (STN). In our framework, a STN can be directly produced from activity graph and mapped into a distance graph. Consistency checking is implemented by a shortest path algorithm (Dechter, Meiri, and Pearl 1991) in the distance graph. It is a popular technique in current research on interval constraints. There would be a subset of activities of an activity graph for each resource. This subset of activities can be expressed with a Resource Temporal Network (RTN). In RTN, resource constraints can be handled by a flaw management mechanism (Laborie 2003), and resolved with envelope-based algorithm or ESTA algorithm (Policella et al. 2009). It is sent to heuristic search process whether there is a solution or not as a feedback.

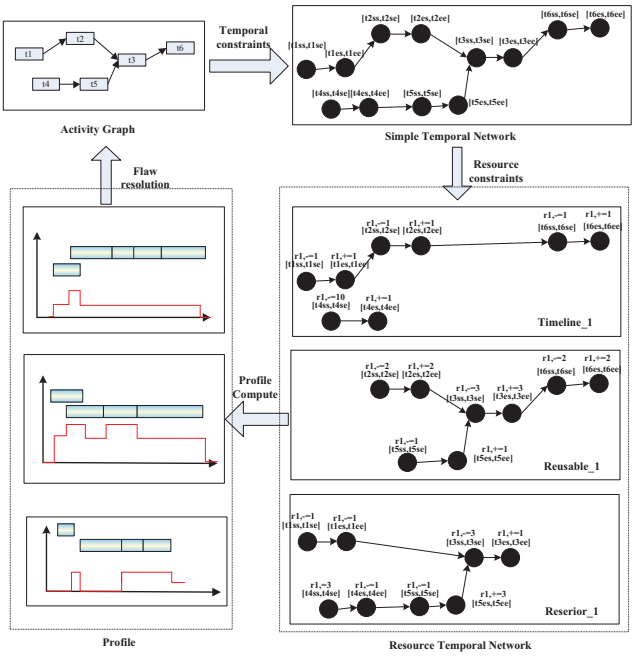


Figure 6: Constraint reasoning process

Thus, we could find suitable solutions by interleaving the heuristic search and constraint reasoning processes. Activity graph bridges the gap of planning and scheduling in solving by hybrid with heuristic search and constraint reasoning.

SMPS: An Implementation

The Space Mission Planning and Scheduling engine (SMPS) is developed as an implementation of our framework. This engine has been designed and implemented as a software module. The engine first receives a problem description with basic concepts, translates the description into a constraint representation, and then solves it by interleaving a heuristic search and constraint reasoning process. This software is not totally stable, but it has already been proved to be suitable for AEOS mission planning problems by a series of simple experiments.

The SMPS engine is written in C# language and implemented on 'Windows' platform. In our implementation,

there are seven components as in Figure 7. A 'Constraint' engine component, based on Cream.Net, is the basic part of the engine. A 'Third Party Functions' component is a encapsulation of dynamic linked library provided by industry departments. A 'Tasks' component expands (Wang et al. 2011) with new activity decomposition model. An 'Activity Graph' component, reference to OMP, bridges the top layer of activities and bottom layer of variables. A 'Solver' component has two parts, heuristic search and constraint reasoning. The heuristic search process consists of a greedy search for high priority tasks and opportunities, and a heuristic selection of assistant activity for preconditions. The constraint reasoning process utilizes Floyd-Warshall to check temporal consistency in STN, sequence selects flaws of RTN to make decision and resolve them with ESTA algorithm. A 'Resources' component utilizes profile to check resource capacity, and transfer third party functions to check special constraints.

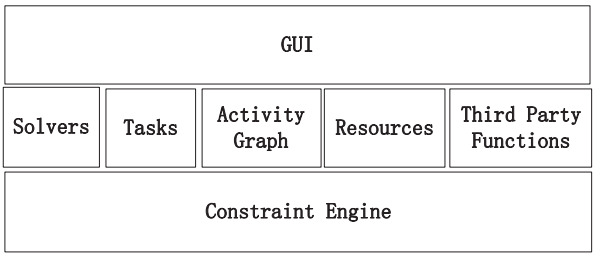


Figure 7: SMPS software components

We selected seven simple examples in AEOS to test our implementation of SMPS. There are different numbers of satellites, ground stations and task requirements in examples, so tasks have different average numbers of imaging opportunities and downloading opportunities. We built plans for the examples with our SMPS. The results (3GB RAM, Intel Core 2, 2.1GHz) are shown in Table 1.

For small instances, our SMPS can find a good solution in limited time. There would be more scheduled tasks if there are much more downloading opportunities between instance2 and instance4. Similarly, there would also be more scheduled tasks if there are more imaging opportunities between instance2 and instance5. However, that is not absolute true, like in instance4 and instance6. The computing time flares up when there are much more tasks to be scheduled, which is because that there is not incremental constraint reasoning yet. We cannot prove that SMPS is a best engine so far, but we can confirm that our framework is suitable for our AEOS problem. And we will test it with much more complex instances.

Conclusions and Future Work

We have presented a constraint-based framework of planning and scheduling for AEOS, introduced basic concepts in the problem description, discussed constraint models and solving techniques. We advocated a top level problem description incorporating planning and scheduling features. In order to gain maximum flexibility in solving problems, we used the constraint representation in constraint model, which allows us to use algorithms from the constraint programming community. In the solving framework, we interleaved heuristic search and constraint reasoning instead of a hybrid of planning and scheduling processes. This framework could handle new challenges of AEOS domain efficiently. For instance, a partial satisfaction problem was described by unrelated tasks and solved by heuristic search, a temporal dependence problem was translated into a constraint model with custom constraints from third party functions and solved by flaw resolution in constraint reasoning, logical constraints were checking by resources and resolved by generating assistant activities and special constraints were defined as custom constraints and checked by resources.

As an extension of RCPSP (Oddi and Rasconi 2009), the framework could also be applied to the RCPSP scheduling. Since any machine scheduling problem can be regarded as a RCPSP problem (Brucker et al. 1999), the framework could also be applied to machine scheduling, e.g. JSSPTW (Oddi et al. 2011). In planning aspect, we have considered logical preconditions of activities, so some planning problems could be handled. Most important thing is that our framework could handle space mission planning problems integrating planning and scheduling features.

In conclusion, our work constitutes a "work-in-progress", several applications are being developed. We are actively considering extensions to support other domains, e.g. geosynchronous earth observation satellites, navigation satellites, deep space observation satellites and even more general domains. We will further give more formal representation and define its semantics. We also would like to consider much more efficient algorithms (Kramer and Smith 2004; Oddi et al. 2011) in solving of our framework.

Acknowledgement

This work was funded by Chinese Natural Science Foundation 71031007, 70801062 and 71071156. This work was also supported by Hunan Provincial Innovation Foundation for Postgraduate and National University of Defense Technology Doctorial Innovation Foundation B110504. The authors would like to thank Thomas, Zhenyu Yang, Yuning Chen and Renjie He for their help in grammar checking, problem understanding and solving.

References

Beaumet, G., and Verfaillie, G. 2007. Estimation of the minimal duration of an attitude change for an autonomous agile earth-observing satellite. *CP 2007* (LNCS 4741):3–17.

Beaumet, G.; Verfaillie, G.; and Charmeau, M. 2007. Autonomous planning for an agile earth-observing satellite.

Bedrax-Weiss, T.; McGann, C.; and Ramakrishnan, S. 2003. Formalizing resources for planning. In *Proceedings of ICAPS-03 Workshops on PDDL*, 7–14.

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.

Table 1 Results of examples computed by our SMPS engine

instance	average no. of imaging opportunities	average no. of downloading opportunities	no. of satellites	no. of ground stations	no. of tasks	no. of scheduled tasks	time (s)
instance1	4	12	2	2	20	18	0.71
instance2	6	12	2	2	20	17	0.73
instance3	5	30	3	3	20	19	1.06
instance4	6	24	3	3	90	27	4.33
instance5	7	12	2	2	100	33	8.75
instance6	7	15	3	3	100	35	10.76
instance7	7	46	4	3	500	80	156

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–94.

Frank, J., and Jonsson, A. 2003. Constraint based attribute and interval planning. *Journal of Constraints* 8(4):339–364.

Frank, J. D.; Clement, B. J.; Chachere, J. M.; Smith, T. B.; and Swanson, K. J. 2011. The challenge of configuring model-based space mission planners. In *Seventh International Workshop on Planning and Scheduling for Space (IWPSS 2011)*.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

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

Grasset-Bourdel, R.; Verfaillie, G.; and Flipo, A. 2011. Building a really executable plan for a constellation of agile earth observation satellites. In *Seventh International Workshop on Planning and Scheduling for Space (IWPSS 2011)*.

Grasset-Bourdel, R. 2010. Interaction between action and motion planning for an agile earth-observating satellite. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

ILog. 2003. *ILOG Scheduler 5.3 User's Manual*.

Kramer, L. A., and Smith, S. F. 2004. Task swapping for schedule improvement, a broader analysis. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling*.

Laborie, P. 2003. Resource temporal networks: Definition and complexity. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, volume 18, 948–953.

Lematre, M.; Verfaillie, G.; Jouhaud, F.; Lachiver, J.-M.; and Bataille, N. 2002. Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology* 6(5):367–381.

Lian, Z., and Xing, L. 2011. A heuristic approach to dynamic rescheduling of AEOS system. In *3nd International Conference on Computer and Network Technology*.

Lian, Z.; Tan, Y.; Xu, Y.; and Li, J. 2011. Static and dynamic models of observation toward earth by agile satellite coverage. In *International Workshop on Planning and Scheduling for Space*.

Muscettola, N. 1994. *HSTS: Integrating Planning and Scheduling*. Intelligent Scheduling.

Oddi, A., and Rasconi, R. 2009. Solving resource-constrained project scheduling problems with time-windows using iterative improvement algorithms. In *Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*.

Oddi, A.; Rasconi, R.; Cesta, A.; and Smith, S. F. 2011. Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times. In *Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*.

OptTime. 2011. Opttime scheduler. available via www.optwise.com.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. F. 2009. Solve-and-robustify: Synthesizing partial order schedules by chaining. *Journal of Scheduling* 12:299–314.

Pralet, C., and Verfaillie, G. 2008. Using constraint networks on timelines to model and solve planning and scheduling problems. In Rintanen, J.; Nebel, B.; Beck, J. C.; and Hansen, E. A., eds., *ICAPS*, 272–279. AAAI.

Wang, P., and Reinelt, G. 2010. A heuristic for an earth observing satellite constellation scheduling problem with download considerations. *Electronic Notes in Discrete Mathematics* 36:711–718.

Wang, P.; Reinelt, G.; Gao, P.; and Tan, Y. 2011. A model, a heuristic and a decision support system to solve the scheduling problem of an earth observing satellite constellation. *Computers & Industrial Engineering* 61:322–335.

Integrated Project Selection and Resource Scheduling of Offshore Oil Well Developments: An Evaluation of CP Models and Heuristic Assumptions

Thiago Serra and **Gilberto Nishioka** and **Fernando J. M. Marcellino**
PETROBRAS – Petróleo Brasileiro S.A., Av. Paulista, 901, 01311-100, São Paulo – SP, Brazil
{thiago.serra, nishioka, fmarcellino}@petrobras.com.br

Abstract

This work aims at introducing and tackling the integrated problem of selecting well development projects and scheduling offshore resources to maximize the short-term production of oil. The importance of incorporating the project selection into the resource scheduling comes from the need for immediate payback that oil companies have due to the expenses for hiring such resources. After all, the return of production can be improved if projects with smaller compensation are avoided. The approach is based on the refinement of a Constraint-Based Scheduling (CBS) model previously reported to tackle the Offshore Resource Scheduling Problem (ORSP), and on the introduction of an alternative model. This latter model relies more heavily on resource constraints in order to take into account the recent developments on their propagation mechanisms. Moreover, accessory constraints are explored to achieve better solutions in short runs. They are intended to increase the domain pruning with heuristic assumptions. The experimental analysis conducted evidences the capability of both models to generate good solutions, a short performance prevalence with the use of additional constraints, and some situations where it is beneficial to use short-run solutions as input to the long runs.

1 Introduction

When oil and gas offshore reservoirs are discovered, one of the most critical bottlenecks that prevents their prompt exploitation is the availability of well development resources. The development of each well relies on a sequence of activities of different types, such as the early stages of well drilling and completion performed by oil rigs, and the later stage of connecting the wells to a producing unit with pipes loaded at harbors and conveyed by pipelay vessels. Besides, a number of technical requirements – such as the range of depth for operation and onboard machinery available – restrains the types of resources capable of performing each activity. Once the corresponding development sequence is concluded, a daily production starts on each well. Thus, the most straightforward alternative to assess the immediate payback of the development resources of an oil company is to maximize the production that can be achieved in a given horizon by scheduling them to develop wells on reservoirs.

The depiction of the Offshore Resource Scheduling Problem (ORSP) has been successively refined since its introduction by Hasle et al. (1996). do Nascimento (2002) showed that it belongs to the class of NP-Hard problems for being as hard as the Job-Shop Scheduling Problem. In other words, it is not known an efficient algorithm to find an optimal solution for the general case of the problem. In the following, Accioly, Marcellino, and Kobayashi (2002), Pereira, Moura, and de Souza (2005), Moura, Pereira, and de Souza (2008), and Serra, Nishioka, and Marcellino (2011) complemented the definition of the problem by including, among other things, resource displacements, onboard inventory and production maximization instead of makespan minimization. Their approach to the problem varied back and forth from Constraint Programming (CP) (Accioly, Marcellino, and Kobayashi 2002; Serra, Nishioka, and Marcellino 2011) to the GRASP metaheuristic (Pereira, Moura, and de Souza 2005; Moura, Pereira, and de Souza 2008). The focus was switched back to CP due to recent developments of the technique, which provided a competitive edge for solving large instances in a short amount of time and leaned the authors of the later work towards using it to approach the problem once more. In addition, Serra, Nishioka, and Marcellino (2011) first approached the issue of assessing the optimality gap of the solutions by means of a straightforward greedy algorithm. Serra, Nishioka, and Marcellino (2012) reduced the optimality gap of that former effort in a benchmark of instances by approaching a relaxation of the problem that corresponds to its prior depiction by Moura, Pereira, and de Souza (2008) with a continuous-time Mixed-Integer Linear Programming (MILP) model. Furthermore, there are several other approaches concerned with similar or related problems in the oil industry. To name but a few, Iyer et al. (1998) addressed a different approach to integrate well development planning and resource scheduling, Glinz and Berumen (2009) considered the resource scheduling of offshore exploratory activities, and Aloise et al. (2006) tackled the onshore scheduling of rigs for well maintenance activities. The ORSP differs from those problems due to the amount of details considered, which also evidences its importance for handling operational decisions in one of the most expensive and return-sensitive processes in the value chain of oil

companies.

The aim of the current refinement of the ORSP is to integrate the project selection as a mean to improve the production payback and to avoid infeasibility dead-ends. Infeasibility can be caused by overconstrained situations in which the available resources are not able to cope with the entire set of activities. Moreover, it is possible to conceive cases in which there are enough resources to do so but a better outcome can be achieved if those resources are not obliged to perform all activities. The problem is approached by two Constraint-Based Scheduling (CBS) models. The first of them is adapted from a model targeting the ORSP without project selection. The second model presents major changes to take the most of the developments related to domain filtering of resource constraints. Additional constraints based on heuristic assumptions are appended to both models for evaluation. They are aimed to reduce the search space in order to provide better solutions in short runs.

The organization of the remainder of the paper is as follows. A definition of the ORSP including project selection is provided in section 2. It is followed by the description of the CP models and additional constraints conceived to tackle the problem in section 3. The conducted experimental evaluation is described in section 4, and the results are discussed in section 5. Final remarks on the achievements and possibilities of future work are presented in section 6.

2 Problem Definition

The Offshore Resources Scheduling Problem (ORSP) can be depicted as a matter of deciding *if*, *when* and *how* to perform each of several activities using a set of resources in a short-term period. Prior to considering project selection as a part of the ORSP, only loading activities used to be considered optional. Each project consists of the development activities of a well that must be performed by rigs and vessels. Each of those activities must be assigned to a resource compatible with its needs. Those resources are required to attend to a large geographical area, where displacements between sites may take a considerable time. Time is represented in days, starting from a date set as 0. For notational convention, resources will be denoted by index i assuming values in the set I, activities by index j assuming values in the set J, and locations by index k assuming values in the set K. Furthermore, $J_W \subseteq J$ denotes the set of activities to be performed on wells, $J_C \subseteq J_W$ the set of connection activities unloading pipes, $K_W \subseteq K$ the set of wells, $J_H = J \setminus J_W$ the set of loading activities, and $K_H = K \setminus K_W$ the set of harbors.

Optimization criteria

We want to maximize the short-term production of the schedule, which is a measure of how much each well would produce since the day its development finishes until a given time horizon H. Each activity j induces a daily production rate pr_j once it is concluded, which is nonzero only in the case of the last activity of each well.

Resource constraints

- Resources like rigs and vessels are unary, meaning that each of them performs at most one activity at a time.
- A resource i can be assigned to perform an activity j if, and only if, $c_{ij} = 1$.
- Only one resource can be assigned to perform an activity on a well at any time.
- If a resource i has to perform consecutive activities j_1 and j_2 on distinct locations, it must be accounted the displacement time $dt_{ik_1k_2}$ between them.
- Each resource i has a contractual period of use, ranging from its release date $rr_i \geq 0$ to its deadline $rd_i \geq rr_i$.
- During that contract period, there are predicted periods of unavailability. Without loss of generality, they can be mapped as activities belonging to a separate set J_M.

Activity constraints

- Well development activities are non-preemptive: they are performed without interruption until their conclusion.
- Each well activity j has to be scheduled between its release date $ar_j \geq 0$ and its deadline $ad_j \geq ar_j$.
- Each activity j is associated with a location loc_j.
- Either all activities in a well are performed or none is.
- Each well activity j requires p_j days to be processed.
- One activity may be preceded by other activities. Let $pc_{j_1j_2} = 1$ if, and only if, activity j_2 is preceded by j_1 and $pd_{j_1j_2}$ be the minimum delay in days between both.
- Some activities may belong to a cluster, in which all activities should be performed by a single resource. Let cl_j denote the index of the cluster of each activity j.

Inventory constraints

- Each resource starts the schedule with an empty inventory.
- Each resource i has a maximum inventory capacity ic_i, which is nonzero only for pipelay vessels.
- Only resources with positive inventory capacity may perform loading activities at harbors.
- Each harbor k may support up to s_k simultaneous loading activities, where $k \in K_W$.
- A loading activity performed by a resource i lasts from mil_i up to mal_i days.
- The increase of the onboard inventory due to a loading activity is proportional to the time it lasts, and a resource i increases its inventory by ic_i after mal_i days.
- If activity j involves a connection, the resource performing it must unload wp_j of pipe weight in the well.
- Loading activities are not subject to clusters or precedence constraints.
- The number of loading activities is not predetermined. However, it is possible to set an upper limit as the product of the number of harbors by the number of connections.

3 Approach

The models designed to tackle the problem are based on the Constraint Programming (CP) technique, or more specifically, on Constraint-Based Scheduling (CBS) solving mechanisms as well as on modeling abstractions involving interval variables. Hence, a short introduction on the subject is given prior to the description of the approach itself.

From CP and CBS to interval variables

Constraint Programming (CP) is defined by Lustig and Puget (2001) as a computer programming technique in which combinatorial problems are formulated using constraints designed to capture problem structure more easily. According to Barták (2001), the strength of the technique comes from the process of constraint propagation. It works by means of filtering algorithms that infer domain reductions involving the decision variables of the problem. They prune the search space before and during the search process. The techniques to ensure the basic consistency levels were first consolidated by Mackworth (1977). Hentenryck, Deville, and Teng (1992) provide a remarkable example on how their performance can be improved if they are specialized. More generally, the competitive edge of CP to approach hard problems is due to how it can handle specificities of application domains, usually by means of global constraints. Bessière and Hentenryck (2003) observe that a global constraint represents a complex but common relation among a set of variables, for which more efficient propagation algorithms are designed. In addition, the techniques beneath the search process vary from the classical backtracking algorithms and their improvements (van Beek 2006) up to adaptive methods (Laborie and Godard 2007) and local search algorithms (Hoos and Stützle 2005). With the advent of algebraic modeling languages supported by CP solvers, Lustig and Puget regarded that the technique was no longer restricted to computer programming experts. Therefore, it was possible to use it in the development of abstract formulations suitable for direct execution in these solvers. In a singular manner, scheduling represents an application domain for which propagation, search and modeling devices evolved noticeably, and thus contributed for a greater use of CP. The interested reader is refered to Dechter (2003) and Baptiste, Le Pape, and Nuijten (2001) for a broader introduction to CP and to Constraint-Based Scheduling (CBS), respectively; and to Baptiste et al. (2006) as well as Barták and Salido (2011) for a slight update on the latter topic.

Interval variables represent an expressive generalization of abstractions such as activities and resources for modeling scheduling problems. The concept as described hereafter was first introduced by Laborie and Rogerie (2008) and further extended by Laborie et al. (2009). Each interval variable depicts an event through a collection of interdependent properties such as its presence, starting date, length and ending date. Those properties can be used to define constraints between pairs of intervals and will be denoted by the following functions over the interval given as argument, respectively: $presence$, $start$, $length$ and end. The consistency of such binary constraints is easily guaranteed by handling Boolean properties as a two-clause Satisfiability Problem (2-SAT) (Laborie and Rogerie 2008), and temporal variables as a Simple Temporal Problem (STS) (Cesta and Oddi 1996). Besides, it is possible to formulate complex relations according to a hierarchical structure imposed by one-to-many constraints such as $alternative$ and $no\text{-}overlap$, as well as cumulative and state functions. The $alternative$ constraint imposes that only one interval in a collection can represent some event and thus occur. Intervals can be grouped into sequences, upon which the $no\text{-}overlap$ constraint can be imposed to state that only one interval can occur at a time. Optionally, a transition function is given to define a setup time between consecutive intervals in the sequence. Intervals can also be used to define cumulative functions over producing and consuming events. In such case, the start and the end of each interval in the cumulative function is associated with step variation coefficients. It is possible to constrain the value of a cumulative function during an interval or on its entire domain. Cumulative functions represent a generalization of cumulative resources, for which many types of consistency have been proposed and refined, and a number of filtering algorithms has been studied. They range from the classical edge-finding algorithm that infers ordering relations between activities in disjunctive resources (Carlier and Pinson 1994) up to recent developments such as energetic reasoning in cumulative resources with activities of variable length (Vilím 2009). Alternatively, state functions can be defined as temporal properties that constrain the execution of conflicting intervals. They can also be subject to transition functions among their states. All the abstractions just mentioned will be used somehow to model the problem hereafter.

Adapting a model previously reported: M_1

The model described in this section is adapted from the one introduced by Serra, Nishioka, and Marcellino (2011). The main difference consists of permitting the absence of well development activities in the current case. Moreover, considering the sparsity of the matrix M, which expresses the combination of activities to resources, separate vectors were used for well and harbor activities. They were indexed by the pairs of resource-activity compatibilities instead. Nevertheless, given that this is more of an implementation issue than a modeling design choice, the previous notation was kept in the remainder of the description for simplicity.

Variables and constraints The first-class depiction by means of interval variables is used to represent vector **a** and matrix **M**. Since each interval of a corresponds to an activity of the schedule, the following domain restrictions apply to those related to well development and resource maintenance activities:

$$start(a_j) \geq ar_j, \qquad \forall j \in J_W \qquad (1)$$
$$end(a_j) \leq ad_j, \qquad \forall j \in J_W \qquad (2)$$
$$length(a_j) = p_j, \qquad \forall j \in J_W \qquad (3)$$
$$presence(a_j) = 1, \qquad \forall j \in J_M \qquad (4)$$

The precedence constraints are directly stated over the intervals of a associated with each activity. The first constraint

below guarantees the chronological order between pairs of activities subject to a precedence relation, while the second one ensures that the presence of the latter interval depends on whether the former is also present. Notice that the latter constraint suffices to handle the project selection, since it is only worth performing an activity of a well if all activities associated with that well are also performed and any resumed development can be removed at post-processing. Besides, tighter constraints involving chains of activities are avoided. The rationale is to facilitate the work of local search algorithms in which the solver could possibly rely, since partial developments are tolerated in this case.

$$end(a_{j_1}) + pd_{j_1 j_2} \leq start(a_{j_2}), \quad \forall j_1 \in J_W, j_2 \in J_W, pc_{j_1 j_2} = 1 \quad (5)$$

$$presence(a_{j_2}) \rightarrow presence(a_{j_1}), \quad \forall j_1 \in J_W, j_2 \in J_W, pc_{j_1 j_2} = 1 \quad (6)$$

Each non-empty cell $m_{ij} \in M$ corresponds to a compatible combination of a resource $i \in I$ and an activity $j \in J$, i.e. $c_{ij} = 1$. Therefore, it is present in a solution if such assignment occurs. The following domain restrictions are imposed according to the type of activity:

$$start(m_{ij}) \geq MAX(rr_i, ar_j), \ \forall i \in I, j \in J_W, c_{ij} = 1 \quad (7)$$

$$end(m_{ij}) \leq MIN(rd_i, ad_j), \quad \forall i \in I, j \in J_W, c_{ij} = 1 \quad (8)$$

$$start(m_{ij}) \geq rr_i, \quad \forall i \in I, j \in J_H, c_{ij} = 1 \quad (9)$$

$$end(m_{ij}) \leq rd_i, \quad \forall i \in I, j \in J_H, c_{ij} = 1 \quad (10)$$

$$length(m_{ij}) \geq mil_i, \quad \forall i \in I, j \in J_H, c_{ij} = 1 \quad (11)$$

$$length(m_{ij}) \leq mal_i, \quad \forall i \in I, j \in J_H, c_{ij} = 1 \quad (12)$$

The vector a and the matrix M are bound by the constraint $alternative$, which states that at most one interval from the j-th column of M occurs and that it corresponds to a_j:

$$alternative(a_j, M_j), \quad \forall j \in J \quad (13)$$

The clustering constraints are represented by logical implications, which force the occurrence of the intervals of all activities of a cluster in the same line of M:

$$presence(m_{i_1 j_1}) \rightarrow \neg\, presence(m_{i_2 j_2}), \quad \forall i_1 \in I, i_2 \in I, j_1 \in J_W, j_2 \in J_W, i_1 \neq i_2, cl_{j_1} = cl_{j_2} \quad (14)$$

The control of concurrency on wells and in the usage of each resource is based on the vectors of sequences **w** and **r**, respectively. For the purposes of this work, each sequence will be denoted as a set of intervals. Those intervals pertain to a in the former case and to M in the latter:

$$w_k = \{a_j \mid j \in J \wedge loc_j = k\}, \quad \forall k \in K_W \quad (15)$$

$$r_i = \{m_{ij} \mid j \in J \wedge c_{ij} = 1\}, \quad \forall i \in I \quad (16)$$

In order to prevent resources from being assigned to more than one activity at a time, the $no\text{-}overlap$ constraint is imposed over each sequence $r_i \in r$. It is accompanied by the transition function $rt_i : K \times K \rightarrow \mathbb{N}$, which defines the displacement time of resource i between consecutive locations:

$$no\text{-}overlap(r_i, rt_i), \quad \forall i \in I \quad (17)$$

A similar constraint over w guarantees that only one activity is performed on each well at a time:

$$no\text{-}overlap(w_k), \quad \forall k \in K_W \quad (18)$$

In the case of harbors, the concurrency on each location is modeled by a different cumulative function of the vector **h**, which is composed of unitary pulses for each loading:

$$h_k = \sum_{j \in J_H : loc_j = k} pulse(a_j, 1), \quad \forall k \in K_H \quad (19)$$

Upper limits to the cumulative functions of h are then imposed over their entire domain:

$$h_k \leq s_k, \quad \forall k \in K_H \quad (20)$$

The control of inventory on each resource is also modeled by means of a cumulative function. There is one element p_i of vector **p** corresponding to each resource $i \in I$. Each cumulative function is composed of positive steps at the end of each loading activity ranging from the smallest to the biggest possible increase of inventory, and negative steps for each activity that unloads inventory:

$$p_i = \sum_{j \in J_H : c_{ij} = 1} stepAtEnd(m_{ij}, 1, ic_i) - \sum_{j \in J_W : c_{ij} = 1} stepAtEnd(m_{ij}, wp_j), \ \forall i \in I \quad (21)$$

Both upper and lower limits are defined to those functions:

$$p_i \geq 0, \quad \forall i \in I \quad (22)$$

$$p_i \leq ic_i, \quad \forall i \in I \quad (23)$$

Moreover, the inventory increase due to each loading is limited by the length of the activity, as follows:

$$heightAtEnd(p_i, m_{ij}) \leq ic_i * \frac{length(m_{ij})}{mal_i}, \quad \forall i \in I, j \in J_H, c_{ij} = 1 \quad (24)$$

Objective function The objective function represents the expected short-term production as the summation of the production triggered by each activity finished in the schedule:

$$\text{max.} \quad \sum_{j \in J_W} MAX(H - end(a_j), 0) * pr_j \quad (25)$$

Introducing a new model to the problem: M_2

The main motivation for introducing an alternative CBS model to the ORSP refers to the unnecessary use of sequences in the model M_1 described above. First, sequences represent not only a set of intervals upon which constraints

can be imposed, but also decision variables that depict an order of such intervals from the domain of all possible permutations (IBM 2010). Therefore, there was a certain concern that the first approach would be prohibitive either in processing time or in memory usage to scale up with respect to the size of the instances to be solved. Second, the simple depiction as a cumulative function would fit most of the needs first addressed by sequences. The only exception is related to the resources displacement time, that can be represented by means of state functions. Finally, an increasingly use of cumulative functions in the modeling could benefit in a larger extent from the recent developments related to the propagation of resource constraints, as mentioned previously.

Changes of the model The vector of sequences w is easily substituted by the vector of cumulative functions **w'**. It handles concurrency on wells in the same way that it is done with harbors:

$$w'_k = \sum_{j \in J : loc_j = k} pulse(a_j, 1), \qquad \forall k \in K_W \quad (26)$$

The introduction of w' requires the replacement of constraint (18) by the following:

$$w'_k \leq 1, \qquad \forall k \in K_W \quad (27)$$

In the case of the vector r, the vector of cumulative functions **r'** and the vector of state functions **r''** are used to represent each a part of what was modeled previously with r:

$$r'_i = \sum_{j \in J : c_{ij} = 1} pulse(m_{ij}, 1), \qquad \forall i \in I \quad (28)$$

$$r''i : \mathbb{N} \rightarrow K \cup \{0\}, \qquad \forall i \in I \quad (29)$$

While r' alone suffices to control the concurrency of use of each resource, r'' is necessary to represent the transition related to resource displacements. Each function $r''_i \in r''$ maps the position of resource i along time. The value 0 is used for transitions and idleness periods.

A similar constraint is imposed over r' to replace constraint (17). It is accompanied by a set of constraints to define the values of r''_i according to the location of i as well as the transition time between different locations:

$$r'_i \leq 1, \qquad \forall i \in I \quad (30)$$

$$r''_i(t) = loc_j, \qquad \forall i \in I, j \in J, \; t \in [start(m_{ij}), end(m_{ij})) \quad (31)$$

$$[r''_i(t_1) = k_1 \wedge r''_i(t_2) = k_2] \rightarrow t_1 + rt_i(k_1, k_2) \leq t_2, \qquad \forall i \in I, k_1 \in K, k_2 \in K, k_1 \neq k_2, t_1 \in \mathbb{N}, t_2 \in \mathbb{N}, t_1 \leq t_2 \quad (32)$$

Adding constraints to both models: M_{1H} & M_{2H}

Some additional constraints are introduced in this section in order to reduce the effort to find solutions with a certain quality in short runs. While it is true that they provide a stronger pruning in the domains of the variables, they also eliminate solutions of all sorts. Nevertheless, they represent the intuitive notion of some end users regarding how a good solution would look like. Therefore, we decided to depict them in order to evaluate the results that can be achieved if they are considered for a while in the solving process.

The first set of constraints considered represent to notion that, if possible, it is preferable to use a single resource wthout interruption to develop a well. It restrains the solutions space in two ways. First, if a resource performs an activity at a well, it also does the following activity if it is compatible. Second, if consecutive activities in the same well are not both connections, they are performed without interruption regardless of whether they use the same resource:

$$presence(m_{ij_1}) \rightarrow presence(m_{ij_2}), \qquad \forall i \in I, j_1 \in J_W, j_2 \in J_W, pc_{j_1 j_2} = 1, c_{ij_1} = 1, c_{ij_2} = 1, loc_{j_1} = loc_{j_2} \quad (33)$$

$$end(a_{j_1}) = start(a_{j_2}), \qquad \forall j_1 \in J_W \setminus J_C, j_2 \in J_W \setminus J_C, pc_{j_1 j_2} = 1, loc_{j_1} = loc_{j_2} \quad (34)$$

The second set of constraints forces the presence of a loading activity for each connection activity, thus forcing each loading activity to carry the pipes of a single well. For such matter, we consider the explicit manipulation of the mapping between the set of connection activities J_C and the set of loading activities J_H. Given a function $ful : J_C \rightarrow J_H$, the constraints below force the presence of one loading activity $j_L \in J_H$ corresponding to the unload of each occurring connection activity $j_U \in J_C$, that they are assigned to the same resource, and constrain the start of the former to be at or after the end of the latter:

$$presence(m_{ij_U}) \rightarrow presence(m_{ij_L}), \qquad \forall i \in I, j_U \in J_C, j_L \in J_H, c_{ij_U} = 1, c_{ij_L} = 1, ful(j_U) = j_L \quad (35)$$

$$start(m_{ij_U}) \geq end(m_{ij_L}), \qquad \forall i \in I, j_U \in J_C, j_L \in J_H, c_{ij_U} = 1, c_{ij_C} = 1, ful(j_U) = j_L \quad (36)$$

4 Experimental Evaluation

The models previously described were tested using a past scenario that comprises the activities to develop 171 wells using 73 resources. Such scenario was used to generate seven instances, all of which using the same set of resources but variable sets of activities. Serra, Nishioka, and Marcellino (2011) made a prior use of those instances. Instance O contains the entire set of activities, which is partitioned approximately into halves for instances $H1$ and $H2$ as well as into quarters for instances $Q1$ to $Q4$. Table 1 summarizes how many activities (Acs.), wells (Wss.), rigs (Rgs.) and vessels (Vss.) each instance (Ins.) has.

Ins.	Q1	Q2	Q3	Q4	H1	H2	O
Acs.	116	118	116	115	231	234	465
Wls.	46	37	45	43	82	89	171
Rgs.	64						
Vss.	9						

Table 1: Characteristics of the tested instances.

The models were implemented in the OPL language and run using the IBM Cplex Studio 12.2 solver. For the run of each model, the time limit was set as one hour in accordance to the end user expectation. Such time was also split between the models M_{1H} and M_1 or M_{2H} and M_2. The output of M_{1H} and M_{2H} are used as input to M_1 and M_2, respectively. The first goal of the hybrid splits was to give to both models an equal time share. But since those additional constraints were intended to provide a short-term benefit, the first share was halved once more. Hence, the following proportions of minutes between each pair of models were tested: $0:60$, $15:45$, $30:30$ and $60:0$. Each instance was run once on each type of model for each time split. The used computer had 4 Dual-Core AMD Opteron 8220 processors, 16 Gb of RAM, and a Linux operating system.

The results of each run are summarized in tables 2 and 3. Along with the solution achieved for each time split of each model type on each instance, the tables also show the upper bounds (U.B.) obtained by Serra, Nishioka, and Marcellino (2012). For the sake of comparability, all values are normalized considering the production of the best solution found for each instance on the experiments of Serra, Nishioka, and Marcellino (2011) as 100. Furthermore, the worst-case optimality gap guaranteed to the best solution found for each instance is presented in table 4. Considering that greater performance variations were observed only for instance O, the progress of the full runs of each model for this instance are presented in figure 1. Such progress is depicted in terms of the best solution found along time.

Ins.	Model	Split	Sol.	Best	U.B.
Q1	M_1	0 : 60	104.00	*	104.4
		15 : 45	104.00	*	
		30 : 30	104.00	*	
		60 : 0	103.94		
	M_2	0 : 60	104.00	*	
		15 : 45	104.00	*	
		30 : 30	104.00	*	
		60 : 0	103.94		
Q2	M_1	0 : 60	113.83	*	113.9
		15 : 45	113.83	*	
		30 : 30	113.83	*	
		60 : 0	113.83	*	
	M_2	0 : 60	113.83	*	
		15 : 45	113.83	*	
		30 : 30	113.83	*	
		60 : 0	113.83	*	
Q3	M_1	0 : 60	100.54	*	100.62
		15 : 45	100.54	*	
		30 : 30	100.53		
		60 : 0	100.50		
	M_2	0 : 60	100.54	*	
		15 : 45	100.54	*	
		30 : 30	100.54	*	
		60 : 0	100.50		
Q4	M_1	0 : 60	103.18		103.53
		15 : 45	103.18		
		30 : 30	103.19		
		60 : 0	103.09		
	M_2	0 : 60	103.20	*	
		15 : 45	103.20	*	
		30 : 30	103.18		
		60 : 0	102.91		

Table 2: Summary of runs for instances Q1 to Q4.

5 Discussion

From the results summarized in tables 2 to 4, we observe a substantial improvement in the best solution found for each instance. According to table 4, the oil production of the best solutions of instances Q1 to Q4, H1 and H2 are all less than 1% smaller than those of optimal solutions. There were varied but expressive improvements in terms of oil production, ranging from 0.54% to 13.83% if they are compared to those achieved by Serra, Nishioka, and Marcellino (2011). In the case of instance O, an improvement of 11.48% was achieved, so that the worst-case optimality gap was reduced to 5.57%. Altogether, such improvements testify once more the capability of CBS to obtain tight and timely results with real-world optimization problems.

The two models presented had similar results in the experimental evaluation. While M_2 performed slightly better in instances Q1 to Q4 and O, it was outperformed by M_1 in instances H1 and H2. However, such difference is explained by the improvements achieved with the use of hybrid splits in the latter case, since the incorporation of additional constraints led to better results in model M_1 if compared to M_2. Nevertheless, stronger conclusions about the comparison of such models would require a large number of independent runs, which could also explore varied levels of propagation for the constraints that differ one model from the other.

The use of heuristic assumptions to impose additional constraints had an increasing impact on the results. On the one hand, the variation is small in table 2 and equal results are attained by varied models. Therefore, it is possible to suppose that the slightly worst results in three out of the four instances for keeping those constraints during all the run translate the impact that they represent in solution quality. On the other hand, from table 3 we observe that better outcomes for instances H1 and H2 were achieved with hybrid splits, but also that the use of such constraints in instance O had a negative effect. Nonetheless, due to the largest gap left for instance O according to table 4, it is also possible to suppose that a bigger time limit could produce different results. Furthermore, from figure 1 it is possible to observe

Ins.	Model	Split	Sol.	Best	U.B.
H1	M_1	0 : 60	104.65		105.00
		15 : 45	104.73	*	
		30 : 30	104.71		
		60 : 0	104.61		
	M_2	0 : 60	104.66		
		15 : 45	104.70		
		30 : 30	104.72		
		60 : 0	104.27		
H2	M_1	0 : 60	105.29		105.91
		15 : 45	105.34	*	
		30 : 30	105.23		
		60 : 0	105.22		
	M_2	0 : 60	105.30		
		15 : 45	105.22		
		30 : 30	105.12		
		60 : 0	104.88		
O	M_1	0 : 60	111.07		118.06
		15 : 45	110.91		
		30 : 30	110.21		
		60 : 0	108.16		
	M_2	0 : 60	111.48	*	
		15 : 45	108.69		
		30 : 30	107.41		
		60 : 0	102.93		

Table 3: Summary of runs for instances H1, H2, and O.

Ins.	Q1	Q2	Q3	Q4	H1	H2	O
Gap	0.38%	0.06%	0.08%	0.32%	0.26%	0.54%	5.57%

Table 4: Worst-case optimality gap of the best solutions found.

that the two models with additional constraints were ahead of their counterparts at some early point in time. Hence, the use of heuristic assumptions is not totally discarded, but it would be necessary to figure out how to calibrate the time split according to the characteristics of each instance.

6 Final Remarks

In this article, we introduced a new element to the Offshore Resource Scheduling Problem (ORSP) as well as discussed relevant issues regarding Constraint-Based Scheduling (CBS) modeling and heuristic assumptions to tackle scheduling problems. The incorporation of project selection to the ORSP enabled to approach the problem in a more realistic and flexible fit. Furthermore, the manner by which the models were presented aimed to help other practitioners when approaching similar problems in at least two ways. First, relevant issues of implementation and subsequent improvements to a model previously described in the literature were presented. Second, an alternative model was also conceived with the intent to leverage CBS strengths according to the trends observed in the technique evolution. Besides, the study on imposing additional constraints based on heuristic assumptions shown the benefit one can have on achieving

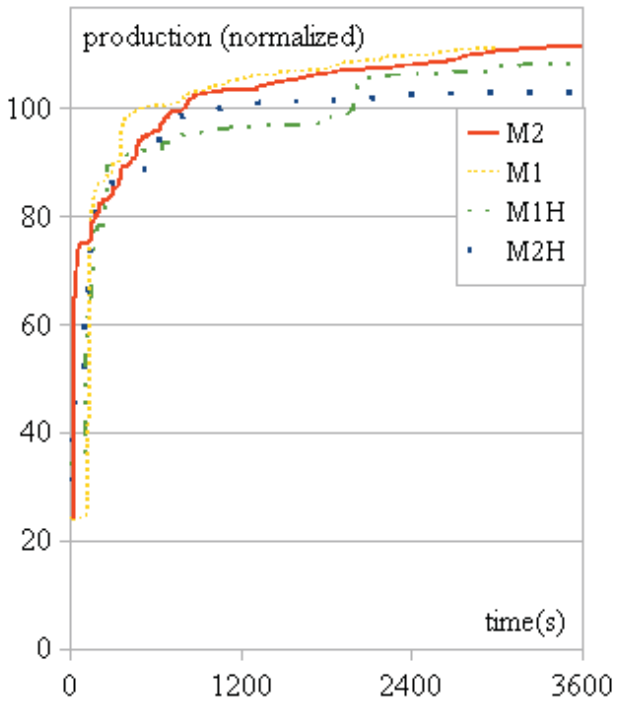


Figure 1: Best solution found along time for the full-time run of each model on instance O.

better solutions earlier in the search, but also the long-term loss if those constraints are kept until the end of the search. Finally, the solutions found with the modification of the implementation as well as with the new model improved considerably the oil production over the results previously reported. In addition, the quality of the results along instances of similar size was much more uniform in the current case. It is also worth to mention that they were obtained with similar hardware and using the same time limit. Hooker (1995) regarded this remark as important, but it is often forgotten in experimental evaluations. Thus, the results evidence the ability of CBS to handle problems like the ORSP. Hence, it becomes clear that this work somewhat concludes the effort on approaching ORSP with CBS for the sort of detail level and size of instances considered.

Future work can be directed either to reduce the remaining worst-case optimality gap or else to incorporate other operational details. As observed by Serra, Nishioka, and Marcellino (2012), it is very likely that a closer gap estimation would require a decomposition approach. Moreover, it is interesting to observe that their approach to assess such gap does not take into account the time required by loading activities. In regard to the introduction of other operational details, it is quite clear that pipes are currently handled almost as a commodity, so that it is possible and maybe desirable to consider some singularities that would differ them by type and availability. Nonetheless, the approach as described in this work and complemented by previous efforts is already at a level of reliance and capability that enables its daily use.

Acknowledgments The authors gratefully acknowledge the company under study for authorizing the publication of the information here present. Furthermore, the opinions and concepts presented are the sole responsibility of the authors.

References

Accioly, R.; Marcellino, F. J. M.; and Kobayashi, H. 2002. Uma aplicação da programação por restrições no escalonamento de atividades em poços de petróleo. In *Proceedings of the 34th Brazilian Symposium on Operations Research*.

Aloise, D. J.; Aloise, D.; Rocha, C. T. M.; Ribeiro, C. C.; Filho, J. C. R.; and Moura, L. S. S. 2006. Scheduling workover rigs for onshore oil production. *Discrete Applied Mathematics* 154:695–702(5).

Baptiste, P.; Laborie, P.; Le Pape, C.; and Nuijten, W. 2006. Constraint-based scheduling and planning. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier. chapter 22.

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

Barták, R., and Salido, M. 2011. *Constraints special issue: Constraint satisfaction for planning and scheduling problems*, volume 16 (3). Springer.

Barták, R. 2001. Theory and practice of constraint propagation. In *Proceedings of the 3rd Workshop on Constraint Programming for Decision and Control (CPDC2001)*, 7–14. Wydavnictvo Pracovni Komputerowej.

Bessière, C., and Hentenryck, P. V. 2003. To be or not to be ... a global constraint. In Rossi, F., ed., *Principles and Practice of Constraint Programming*, volume 2823 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 789–794.

Carlier, J., and Pinson, E. 1994. Adjustment of heads and tails for the job-shop problem. *European Jornal of Operational Research (EJOR)* 78:146–161.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96)*.

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

do Nascimento, J. M. 2002. Ferramentas computacionais híbridas para a otimização da produção de petróleo em águas profundas. Master's thesis, Universidade Estadual de Campinas (Unicamp), Brazil.

Glinz, I., and Berumen, L. 2009. Optimization model for an oil well drilling program: Mexico case. *Oil and Gas Business* 1.

Hasle, G.; Haut, R.; Johansen, B.; and Ølberg, T. 1996. Well activity scheduling - an application of constraint reasoning. In *Artificial Intelligence in the Petroleum Industry: Symbolic and Computational Applications II*. Technip. 209–228.

Hentenryck, P. V.; Deville, Y.; and Teng, C. M. 1992. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence* 57.

Hooker, J. 1995. Testing heuristics: We have it all wrong. *Journal of Heuristics* 1:33–42.

Hoos, H. H., and Stützle, T. 2005. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann.

IBM. 2010. *ILOG CPLEX Optimization Studio 12.2 documentation for ODM Enterprise*.

Iyer, R.; Grossmann, I.; Vasantharajan, S.; and Cullick, A. 1998. Optimal planning and scheduling of offshore oil field infrastructure investment and operations. *Industrial & Engineering Chemistry Research* 37:1380–1397.

Laborie, P., and Godard, D. 2007. Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2007)*, 276–284.

Laborie, P., and Rogerie, J. 2008. Reasoning with conditional time-intervals. In *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference (FLAIRS 2008)*.

Laborie, P.; Rogerie, J.; Shaw, P.; and Vilim, P. 2009. Reasoning with conditional time-intervals, part ii: An algebraical model for resources. In *Proceedings of the 22st International Florida Artificial Intelligence Research Society Conference (FLAIRS 2009)*.

Lustig, I. J., and Puget, J. F. 2001. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces* 31:29–53(6).

Mackworth, A. 1977. Consistency in networks of relations. *Artificial Intelligence* 8:99118(1).

Moura, A. V.; Pereira, R. A.; and de Souza, C. C. 2008. Scheduling activities at oil wells with resource displacement. *International Transactions in Operational Research* 15(25):659–683.

Pereira, R. A.; Moura, A. V.; and de Souza, C. C. 2005. Comparative experiments with GRASP and constraint programming for the oil well drilling problem. In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms*, 328–340. Springer.

Serra, T.; Nishioka, G.; and Marcellino, F. J. M. 2011. A constraint-based scheduling of offshore well development activities with inventory management. In *Proceedings of the 43rd Brazilian Symposium on Operations Research*.

Serra, T.; Nishioka, G.; and Marcellino, F. J. M. 2012. On estimating the return of resource aquisitions through scheduling: An evaluation of continuous-time milp models to approach the development of offshore oil wells. In *Proceedings of the 6th Scheduling and Planning Applications Workshop*. To be held.

van Beek, P. 2006. Backtracking search algorithms. In Rossi, F.; van Beek, P.; and Walsh, T., eds., *Handbook of Constraint Programming*. Elsevier. chapter 4.

Vilím, P. 2009. Max energy filtering algorithm for discrete cumulative resources. In *CPAIOR '09: Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 294–308. Springer-Verlag.