Report Number: WUCSE-2002-13

2002-05-23

Modeling and Solving a Resource Allocation Problem with Soft Constraint Techniques

Weixiong Zhang

We study a resource allocation problem, which is a central piece of a real-world crew scheduling problem. We first formulate the problem as a hybrid soft constraint satisfaction and optimization problem and show that its worst-case complexity is NP-complete. We then propose and study a set of decision and optimization modeling schemes for the problem. We consider the expressiveness of these modeling schemes for the problem. We consider the expressiveness of these modeling methods. Specifically, we experimentally investigate how these modeling schemes interplay with the best existing systematic search and local search methods. Our experimental results show that soft... **Read complete abstract on page 2.**

Modeling and Solving a Resource Allocation Problem with Soft Constraint Techniques

Weixiong Zhang

Complete Abstract:

We study a resource allocation problem, which is a central piece of a real-world crew scheduling problem. We first formulate the problem as a hybrid soft constraint satisfaction and optimization problem and show that its worst-case complexity is NP-complete. We then propose and study a set of decision and optimization modeling schemes for the problem. We consider the expressiveness of these modeling schemes for the problem. We consider the expressiveness of these modeling methods. Specifically, we experimentally investigate how these modeling schemes interplay with the best existing systematic search and local search methods. Our experimental results show that soft constraint techniques can be effective on large resource allocation problem instances, and an optimization approach is more efficient than a model checking approach based on decision models.

Department of Computer Science & Engineering

Washington University in St. Louis
School of Engineering & Applied Science

2002-13

Modeling and Solving a Resource Allocation Problem with Soft Constraint Techniques

Authors: Zhang, Weixiong

May 23, 2002

Abstract: We study a resource allocation problem, which is a central piece of a real-world crew scheduling problem. We first formulate the problem as a hybrid soft constraint satisfaction and optimization problem and show that its worst-case complexity is NP-complete. We then propose and study a set of decision and optimization modeling schemes for the problem. We consider the expressiveness of these modeling methods. Specifically, we experimentally investigate how these modeling schemes interplay with the best existing systematic search and local search methods. Our experimental results show that soft constraint techniques can be effective on large resource allocation problem instances, and an optimization approach is more efficient than a model checking approach based on decision models.

Modeling and Solving a Resource Allocation Problem with Soft Constraint Techniques*

Weixiong Zhang

Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130
zhang@cs.wustl.edu

Technical Report: WUCS-2002-13
Date: May 23, 2002

Abstract. We study a resource allocation problem, which is a central piece of a real-world crew scheduling problem. We first formulate the problem as a hybrid soft constraint satisfaction and optimization problem and show that its worst-case complexity is NP-complete. We then propose and study a set of decision and optimization modeling schemes for the problem. We consider the expressiveness of these modeling methods. Specifically, we experimentally investigate how these modeling schemes interplay with the best existing systematic search and local search methods. Our experimental results show that soft constraint techniques can be effective on large resource allocation problem instances, and an optimization approach is more efficient than a model checking approach based on decision models.

1 Introduction

Resource allocation problems usually lie at the core of many real-world scheduling and planning problems. The properties of an underlying resource allocation problem can help characterize a scheduling problem. If the resource allocation problem is a critical piece of the scheduling problem, the complexity of the former will dominate the complexity of the latter. If the resource allocation problem is difficult, the scheduling problem is doomed to be hard as well.

The first steps toward understanding a resource allocation problem consist of formulating and modeling the problem. Modeling is one of the central themes of AI and a critical step of problem solving. A good model can usually lend itself to an efficient problem-solving strategy. However, modeling is difficult because there are many affecting factors, and no general guidelines exist on which factors must be taken into account

* This research was funded in part by NSF Grants IIS-0196057 and EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. Thanks to USC/ISI Camera group for bringing to us the scheduling problem, to Alejandro Bugacov for many helpful discussions, and to Peng Wang and Xiaotao Zhang for experimentation. Some preliminary results of this research were presented at *CP2001 Workshop on Modeling and Solving Problems with Soft Constraints*.

		R_1	R_2	R_3
T_1	$Q_{1,1}$	0	1	1
	$Q_{1,2}$	1	0	0
T_2	$Q_{2,1}$	1	1	0
	$Q_{2,2}$	1	1	0

Table 1. A simple bundled, exclusive resource allocation problem.

in a modeling process. Therefore, developing a good model remains largely an art, depending on experience and taste. Nevertheless, for a given problem, different modeling techniques can be studied, and their expressiveness and complexity can be compared.

In this paper, we consider a resource allocation problem, which resides at the center of a complex, real-world task scheduling problem [3]. We call the problem *bundled, exclusive resource allocation problem*, or BERAP for short. Briefly, the problem is to allocate a set of shared resources to satisfy the resource requirements of a set of tasks. The difficulties of the problem stem from the restrictions that a task is not fulfilled until all its resource requirements are met and that a resource can only be used to meet one resource requirement.

Our objectives of this research are multifold. The first is to solve the resource allocation problem using soft constraint techniques [1, 9]. The second is to analyze the complexity of the problem. The third objective is to develop constraint models for the problem and analyze their representational power. The final and most important objective is to understand the possible interplay between constraint models and problem-solving strategies and search algorithms.

We proceed as follows. We first describe the problem in Section 2 and analyze its worst-case complexity in Section 3. We discuss the main modeling choices in Section 4 and hard and soft constraints in Section 5. We develop constraint models Sections 6 and 6. These models are experimentally evaluated by systematic and nonsystematic search algorithms in Section 8. We discuss related work in Section 9, and finally conclude in Section 10.

2 Bundled, Exclusive Resource Allocation Problem

The resource allocation problem was originated from a screw scheduling problem [3]. We are given a set of t tasks, $\mathcal{T} = \{T_1, T_2, \cdots, T_t\}$, and a set of r resources, $\mathcal{R} = \{R_1, R_2, \cdots, R_r\}$. Each task requires a certain number of resources in order to execute, which we call resource requirements. Each resource can only be allocate to one resource requirement, and a resource requirement can be met by having one desirable resource allocated to it. We denote the q_i resource requirements of task T_i by $\mathcal{Q}_i = \{Q_{i,1}, Q_{i,2}, \cdots, Q_{i,q_i}\}$. Table 1 shows a small example of resource requirements of two tasks over three resources. An entity of 1 (0) in the table means that a resource can (cannot) be allocated to the corresponding requirement. In general, the available resources may not be sufficient to fulfill every task; and a task carries a penalty, called *task penalty*, if not scheduled. The resource allocation problem is to allocate the resources to the tasks so that the overall penalty of unfulfilled tasks is minimized, an optimization problem. If all tasks have equal penalties, it is equivalent to fulfill the maximal number of tasks.

Compared to some other resource allocation problems, for instances the permutation problems considered in [12, 14], our problem has a unique, small structure embedded within a task. A task can be scheduled if and only if all its resource requirements are met. We call this feature *bundled resource requirement*. Furthermore, a pair of resource requirements have an exclusive resource contention in that a resource acquired by one requirement cannot be allocated to the others. We call this feature *exclusive resource contention*. Overall, we call the problem *bundled, exclusive resource allocation problem*, or *BERAP* for short.

3 Computational Complexity

We now show that BERAP is NP-hard [5]. To this end, we prove that a decision version of the problem is NP-complete [5]. A simple, special decision version of BERAP is the following. Given a set of tasks, each of which has a set of resource requirements, decide if at least k tasks can be fulfilled. Here we simply consider every task having a penalty one if it is not fulfilled.

Theorem 1. *BERAP with more than two resource requirements per task is NP-complete.*

Proof: We show the above decision version of BERAP is NP-complete. We reduce a NP-complete set packing problem [5] to this decision problem. Given a collection S of finite sets and a positive integer $K \leq |S|$, set packing is to decide if S contains at least K mutually disjoint subsets. Formally, it is to decide if there exists $S' \subseteq S$ such that $|S'| \geq K$ and for all $S_1 \in S'$ and $S_2 \in S'$, $S_1 \cap S_2 = \emptyset$. The problem is NP-complete when every subset $S_i \in S$ has more than two elements. We now reduce an NP-complete set packing problem to our decision BERAP. We map all the elements in the subsets of a set packing problem instance to the resources of BERAP, each subset of the set packing instance to a task of BERAP, and an element in the subset to a resource requirement of the respective task. In other words, the total number of tasks is the number of subsets $|S|$, the number of resources is the number of distinct elements in all subsets of S, and the number of resource requirements of a task is the number of elements in the corresponding subset. Given $K \leq |S|$, the constructed BERAP is to decide if at least K tasks can be fulfilled. Clearly, a solution to the BERAP is also a solution to the original set packing problem. □

4 Modeling Considerations and Choices

Three modeling decisions should be made when building a constraint model. The first is to choose variables and values. For instance, we may use resource requirements as variables and desirable resources as their values, or vice versa. A good choice of variables and values seems to bear upon the representational power of the resulting models and the search algorithm use. There may exist soft constraints that is hidden in a model built from one modeling choice which may be explicit in another model from a different choice. There is a tradeoff between the expressiveness of a model and the efficiency of a search algorithm. One of our objectives of this research is to understand the expressiveness of different design schemes and their impact on the performance of search algorithms.

	Optimization	Decision
general variables	a COP model	a set of CSP models
Boolean variables	a MAX-SAT model	a set of SAT models

Table 2. Four modeling choices for soft constraint optimization problems.

The second and subtle modeling decision is to choose the variable types. We may use regular variables with large domains to develop a general CSP or COP model, or Boolean variables to build a Boolean satisfiability (SAT) or maximum Boolean satisfiability (MAX-SAT) model. This consideration is inspired by the success of Davis-Putnam-Longmann-Loveland algorithm [2] and local search for solving SAT [11, 10]. We will compare these choices in this research.

The third and difficult decision is to choose between a CSP model and a COP model. Using a CSP model follows the spirit of model checking [7]. However, many real-world applications, including the scheduling problem where our BERAP resides, are optimization problems. Taking a CSP approach will force one to solve a set of CSP models, each of which is for finding a solution of a prescribed quality, such as a schedule with a certain number of tasks to be fulfilled. Such CSP models can be built and used to carry out a binary search in the space of solution quality to reduce complexity. If a CSP model can be solved efficiently and the number of optimal solutions is limited, the CSP approach may be a good choice. For instance, when the goal for a BEREP is to fulfill the maximal number of tasks possible, the CSP approach may be feasible when such maximal number of possible tasks is relatively small. A serious problem with this CSP-based, model checking approach is that it does not seem to be applicable when tasks have different penalties if not fulfilled and the overall goal is to minimize the total penalty of unscheduled tasks. When the number of optimal solutions is large or the goal is to minimize the total penalty of unfulfilled tasks, solving the problem directly by a COP approach may be more tractable. Nevertheless, in this research we consider the CSP-based, model-checking approach and study how it compares with a direct optimization approach.

As summarized in Table 2, we have four modeling schemes, based on the decision of variables and values as well as problem-solving strategies.

5 Hard and Soft Constraints

Hard constraints is the ones that must be protected; a violation to one of them is forbidden or will incur a prohibitively high penalty. In contrast, soft constraints are the ones that may be violated; a violation to one of them is not fatal but will degrade the quality of a solution [1]. Therefore, to distinct soft and hard constraints in a constraint model, we give them different penalties. The penalty for violating a hard constraint should be set to such a prohibitively high value that a search algorithm would rather violate all soft constraints than violate the hard constraint. Thus, the penalty for a hard constraint should be at least as large as the total penalty of all soft constraints plus one.

Not all tasks of a BERAP can be fulfilled if resources are insufficient. We use a Boolean variable T_i to represent task T_i. Variable $T_i = 1$ if task T_i is fulfilled, $T_i = 0$

otherwise. We then introduce a constraint $(T_i = 1)$, call a *task constraint*, to specify fulfilling task T_i. As a task constraint may be violated, so that it is a soft constraint. We use $\mathcal{C}_t = \bigwedge_{i=1}^{t}(T_i = 1)$ or $\mathcal{C}_t = \bigwedge_{i=1}^{t} T_i$ to represent the set of soft task constraints. The penalty for an unsatisfied task constraint should be at least equal to the penalty of the corresponding task.

The two eminent features of BERAP, *bundled resource requirements* and *exclusive resource contentions*, make the problem difficult to model and solve. They impose hard restrictions on how resources can and should be allocated. We use hard constraints to represent such restrictions.

To facilitate our discussion, we use $\mathcal{C}_b$ and $\mathcal{C}_e$ to short handed the set of *bundled constraints* and $\mathcal{C}_e$ the set of *exclusion constraints*, respectively. Therefore, the overall constraints of an BERAP can be written as $\mathcal{C} = \mathcal{C}_t \bigwedge \mathcal{C}_b \bigwedge \mathcal{C}_e$. How to properly model the hard constraints in $\mathcal{C}_b$ and $\mathcal{C}_e$ and define a penalty function for each of them are the main focus of our modeling effort.

6 Modeling in Soft Constraint Optimization

Models in constraint optimization are called MAX-SAT models if Boolean variable are used, or COP models otherwise. We discuss these two types of models in turn. We will use our toy example in Table 1 to illustrate the resulting models.

6.1 COP models

Different choices of variable/value pairs lead to different models.

Model 1: Requirements as variables Our first attempt is to cast resource requirements as variables and their desirable resources as values. We further introduce a NULL value, denoted as $\emptyset$, into a variable domain to represent the case when the requirement has no resource allocated. For instance, the domain of requirement variable $Q_{1,1}$ of Table 1 is $\{\emptyset, R_2, R_3\}$.

We now consider the hard, bundle constraints $\mathcal{C}_b$. We introduce a hard constraint to specify if a task constraint is satisfied, i.e., the corresponding task variable has value 1, then a requirement variable associated with the task cannot have a NULL value. For instance, when task variable $T_1 = 1$ in Table 1, the requirement variable $Q_{1,1} \neq \emptyset$. We thus write a hard constraint $((T_1 = 0) \vee (Q_{1,1} \neq \emptyset))$. Spelling out all such hard constraints for all tasks, we then have

$$\mathcal{C}_b = \bigwedge_{i=1}^{t} \mathcal{C}_b(T_i); \quad \mathcal{C}_b(T_i) = \bigwedge_{j=1}^{q_i} ((T_i = 0) \vee (Q_{i,j} \neq \emptyset)). \tag{1}$$

We now turn to exclusion constraints, which prevent a resource to be allocated to multiple requirements. We use a constraint to exclude the possibility that two requirement variables have the same, non-NULL value. In our example, to exclude allocating resource R_1 to $Q_{1,2}$ and $Q_{2,1}$ at the same time, we write a hard constraint $((Q_{1,2} \neq R_1) \vee (Q_{2,1} \neq R_1))$. For all resources, we write,

$$\mathcal{C}_e = \bigwedge_{k=1}^{r} \mathcal{C}_e(R_k); \quad \mathcal{C}_e(R_k) = \bigwedge((Q_{i_1,j_1} \neq R_k) \vee (Q_{i_2,j_2} \neq R_k)). \tag{2}$$

where $1 \leq i_1 \leq n; i_1 \leq i_2 \leq n; 1 \leq j_1 \leq q_{i_1}; 1 \leq j_2 \leq q_{i_2}; j_1 \neq j_2$ if $i_1 = i_2$. Note that requirement variable Q_{i_1,j_1} may be the same as Q_{i_2,j_2}.

To illustrate, we have a constraint model for our example problem as follows.

$$\begin{cases} \mathcal{C}_t = (T_1 = 1), (T_2 = 1) \\ \mathcal{C}_b = \begin{cases} ((T_1 = 0) \vee (Q_{1,1} \neq \emptyset)), ((T_1 = 0) \vee (Q_{1,2} \neq \emptyset)) \\ ((T_2 = 0) \vee (Q_{2,1} \neq \emptyset)), ((T_2 = 0) \vee (Q_{2,2} \neq \emptyset)) \end{cases} \\ \mathcal{C}_e = \begin{cases} ((Q_{1,2} \neq R_1) \vee (Q_{2,1} \neq R_1)), ((Q_{1,2} \neq R_1) \vee (Q_{2,2} \neq R_1)) \\ ((Q_{2,1} \neq R_1) \vee (Q_{2,2} \neq R_1)), ((Q_{1,1} \neq R_2) \vee (Q_{2,1} \neq R_2)) \\ ((Q_{1,1} \neq R_2) \vee (Q_{2,2} \neq R_2)), ((Q_{2,1} \neq R_2) \vee (Q_{2,2} \neq R_2)) \end{cases} \end{cases}$$

Model 2: Resource as variables Our second method is to use resources as variables and resource requirements as their values. Take the first resource R_1 of Table 1 as an example. Resource variable R_1 has a domain $\{Q_{1,2}, Q_{2,1}, Q_{2,2}\}$.

To define the bundle constraints, we introduce a hard constraint for each resource requirement to specify that the associated task cannot be fulfilled without the requirement unsatisfied. In our example for instance, to specify that when T_1 is fulfilled its requirement $Q_{1,1}$ must be satisfied, we write a hard constraint $((T_1 = 0) \vee (R_2 = Q_{1,1}) \vee (R_3 = Q_{1,1}))$. We thus write all bundled constraints as follows

$$\mathcal{C}_b = \bigwedge_{i=1}^{t} \mathcal{C}_b(T_i); \quad \mathcal{C}_b(T_i) = \bigwedge_{j=1}^{q_i} ((T_i = 0) \bigvee_{k=1}^{r} (R_k = Q_{i,j})) \tag{3}$$

Fortunately, the exclusion constraints disappear automatically in this model, because a resource variable can have only one value at a time. The size of this model is typically smaller than that of Model 1.

For our working example, the constraints of the model can be written as

$$\begin{cases} \mathcal{C}_t = (T_1 = 1), (T_2 = 1) \\ \mathcal{C}_b = \begin{cases} ((T_1 = 0) \vee (R_2 = Q_{1,1}) \vee (R_3 = Q_{1,1})), ((T_1 = 0) \vee (R_1 = Q_{1,2})) \\ ((T_2 = 0) \vee (R_1 = Q_{2,1}) \vee (R_2 = Q_{2,1})), ((T_2 = 0) \vee (R_1 = Q_{2,2}) \vee (R_2 = Q_{2,2})) \end{cases} \end{cases}$$

Model 3: Resource as variables, a more explicit model In Model 2, a requirement may grab more than one resource. This happens when multiple resource variables have the same requirement as their values. Such a scenario is legitimate, but may be wasteful when resources are scarce. If we exclude such an undesirable, redundant resource allocation, the resulting model will be more expressive and larger than the original version as more hidden constraints will be brought to bear. Whether or not an expressive model is more effective than a less expressive model is a question to be examined closely, a topic of Section 8.

To make Model 2 explicit, we introduce constraints to prevent a requirement to have more than one resource. For example, we exclude the requirement $Q_{1,1}$ has both resources R_2 and R_3 using a constraint $((R_2 \neq Q_{1,1}) \vee (R_3 \neq Q_{1,1}))$. In general, we have to consider each possible pair of desirable resources of a requirement, and rewrite the bundle constraints as

$$\begin{cases} \mathcal{C}_b = \bigwedge_{i=1}^{t} \mathcal{C}_b(T_i) \bigwedge_{i=1}^{t} \mathcal{C}'_b(T_i) \\ \mathcal{C}_b(T_i) = \bigwedge_{j=1}^{q_i} ((T_i = 0) \bigvee_{k=1}^{r} (R_k = Q_{i,j})) \\ \mathcal{C}'_b(T_i) = \bigwedge_{1 \leq k_1 \leq r; k_1 < k_2 \leq r; 1 \leq j \leq q_i} (R_{k_1} \neq Q_{i,j} \vee (R_{k_2} \neq Q_{i,j})). \end{cases} \tag{4}$$

The new constraints are soft, violating them can only result in resource wast. Furthermore, comparing to the soft task constraints, the soft bundled constrains are less important and should thus carry smaller penalties. We may give each soft bundled constraint a penalty one, and each soft task constraint a penalty equal to the total number of soft bundled constraints plus the penalty of the task. The penalty of a hard constraint is then adjusted accordingly.

For our working example, the new soft bundled constraints are,

$$\mathcal{C}_b' = \{ ((R_2 \neq Q_{1,1}) \vee (R_3 \neq Q_{1,1})), ((R_1 \neq Q_{2,1}) \vee (R_2 \neq Q_{2,1})), ((R_1 \neq Q_{2,2}) \vee (R_2 \neq Q_{2,2}))$$

6.2 MAX-SAT models

In MAX-SAT models, the task variables are Boolean. Therefore, task constraint $(T_i = 1)$ is simply (T_i).

Model 4: Requirements as Boolean variables This is parallel to Model 1. The only difference is that instead of one variable per resource requirement, we now have a set of Boolean variables per requirement. Consider a requirement variable $Q_{i,j}$ with domain $\{R_{k_1}, R_{k_2}, \cdots, R_{k_v}, \emptyset\}$. We introduce v Boolean requirement variables, $\{Q_{i,j,k_1}, Q_{i,j,k_2}, \cdots, Q_{i,j,k_v}\}$, where Boolean variable Q_{i,j,k_l} corresponds to resource R_{k_l}. Variable Q_{i,j,k_l} has value T if resource R_{k_l} is assigned to $Q_{i,j}$, and value F otherwise. With Boolean variables, we rewrite the hard bundled constraints as

$$\begin{cases} \mathcal{C}_b = \bigwedge_{i=1}^{t} \mathcal{C}_b(T_i), \\ \mathcal{C}_b(T_i) = \bigwedge_{j=1}^{q_i} (\neg T_i \bigvee_{k=1}^{r} Q_{i,j,k}). \end{cases} \tag{5}$$

Similarly, we update the exclusion constraints to

$$\begin{cases} \mathcal{C}_e = \bigwedge_{k=1}^{r} \mathcal{C}_e(R_k), \\ \mathcal{C}_e(R_k) = \bigwedge_{1 \leq i_1 \leq n; i_1 \leq i_2 \leq n; 1 \leq j_1 \leq q_{i_1}; 1 \leq j_2 \leq q_{i_2}; j_1 \neq j2 \text{ if } i_1 = i_2} \\ \qquad (\neg Q_{i_1,j_1,k} \vee \neg Q_{i_2,j_2,k}) \end{cases} \tag{6}$$

For our working example, the MAX-SAT model is given as follows.

$$\begin{cases} \mathcal{C}_t = T_1, T_2 \\ \mathcal{C}_b = (\neg T_1 \vee Q_{1,1,2} \vee Q_{1,1,3}), (\neg T_1 \vee Q_{1,2,1}), (\neg T_2 \vee Q_{2,1,1} \vee Q_{2,1,2}), (\neg T_2 \vee Q_{2,2,1} \vee Q_{2,2,2}) \\ \mathcal{C}_e = \begin{cases} (\neg Q_{1,2,1} \vee \neg Q_{2,1,1}), (\neg Q_{1,2,1} \vee \neg Q_{2,2,1}), (\neg Q_{2,1,1} \vee \neg Q_{2,2,1}) \\ (\neg Q_{1,1,2} \vee \neg Q_{2,1,2}), (\neg Q_{1,1,2} \vee \neg Q_{2,2,2}), (\neg Q_{2,1,2} \vee \neg Q_{2,2,2}) \end{cases} \end{cases}$$

Model 5: Requirements as Boolean variables, a more explicit model The use of Boolean variables in Model 4 introduces additional constraints, i.e., multiple Boolean requirement variables for a resource requirement may have value T at the same time, meaning that the requirement holds multiple resources, a similar problem that Model 3 attempts to correct over Model 2. Similarly, we introduce additional constraints to prevent this from happening, making some hidden soft constraints explicit. One such mutually exclusive constraint is $(\neg Q_{i,j,k_1} \vee \neg Q_{i,j,k_2})$, meaning that Boolean variables

Q_{i,j,k_1} and Q_{i,j,k_2} should not have value T simultaneously. Specifically, we have the following soft constraints:

$$\begin{cases} \mathcal{C}_b = \bigwedge_{i=1}^{t} \mathcal{C}_b(T_i) \bigwedge_{i=1}^{t} \mathcal{C}'_b(T_i) \\ \mathcal{C}_b(T_i) = \bigwedge_{j=1}^{q_i} (\neg T_i \bigvee_{k=1}^{r} Q_{i,j,k}) \\ \mathcal{C}'_b(T_i) = \bigwedge_{1 \le j \le q_i; 1 \le k_1 \le r; k_1 < k_2 \le r} (\neg Q_{i,j,k_1} \vee \neg Q_{i,j,k_2}). \end{cases} \tag{7}$$

It is interesting to note that MAX-SAT models corresponding to COP Model 2 and Model 3 collapse into the same model, MAX-SAT Model 5. A direct argument is left as an exercise for the reader.

The model for our example is then Model 4 plus the following,

$$\mathcal{C}'_b = (\neg Q_{1,1,2} \vee \neg Q_{1,1,3}), (\neg Q_{2,1,1} \vee \neg Q_{2,1,2}), (\neg Q_{2,2,1} \vee \neg Q_{2,2,2})$$

7 Modeling in Soft Constraint Satisfaction

The objective here is to create a set of models, each of which specifies at least a fixed number k of tasks to be fulfilled. These models are then checked to verify if at least k tasks are indeed satisfiable. The overall process searches for the maximal number of tasks to be scheduled.

7.1 CSP models

The key to building a CSP model for an optimization problem is to introduce constraints to represent the goal of fulfilling at least a fixed number of tasks. Specifying at least k tasks to be turned on is realized by a set of dummy variables, an idea proposed in [3]. We introduce n dummy variables, one for each task. The domain of V_i is $\{0, 1, 2, \cdots, k\}$, where k is the number of tasks to be turned on. The value 0 in the domain is special. A dummy variable is turned off if its value is 0, or turned on otherwise. We then connect V_i with task variable T_i by hard constraints specifying that task T_i can be fulfilled if and only if its dummy variable V_i is turned on. That is, we write hard constraints, $((T_i = 0) \vee (V_i \neq 0))$ and $((T_i = 1) \vee (V_i = 0))$. The first constraint means that when task T_i is fulfilled, i.e., $(T_i = 0)$, its associated variable must be on, i.e., $V_i \neq 0$. The second constraint specifies the case when task T_i is unfulfilled, its associated dummy variable cannot be turned on.

Furthermore, we introduce constraints to turn on at least k dummy variables so as to fulfill at least k tasks. We use constraints to specify that one of V_i must take value j, for $j = 1, 2, \cdots, k$. That is, we have a total of k hard constraints $((V_1 = j) \vee (V_2 = j) \vee \cdots \vee (V_n = j))$, for $j = 1, 2, \cdots, k$. We call all these hard constraints *CSP constraints*, denoted as $\mathcal{C}_{CSP}^{(k)}$, In short, we write

$$\mathcal{C}_{CSP}^{(k)} = \begin{cases} \bigwedge_{i=1}^{t} ((T_i = 1) \vee (V_i = 0)) \\ \bigwedge_{i=1}^{t} ((T_i = 0) \vee (V_i \neq 0)) \\ \bigwedge_{j=1}^{k} (\bigvee_{i=1}^{t} (V_i = j)) \end{cases}$$

To turn on 2 task in our example of Table 1, the CSP constraints are,

$$\mathcal{C}_{CSP}^{(2)} = \begin{cases} ((T_1 = 1) \vee (V_1 = 0)), ((T_1 = 0) \vee (V_1 \neq 0)), ((T_2 = 1) \vee (V_2 = 0)) \\ ((T_2 = 0) \vee (V_2 \neq 0)), ((V_1 = 1) \vee (V_2 = 1)), ((V_1 = 2) \vee (V_2 = 2)) \end{cases}$$

One of the COP models, Model 1, Model 2 or Model 3 presented in Section 6 can be combined with a set of CSP constraints $\mathcal{C}_{CSP}^{(k)}$ to form a CSP model. We can therefore derive three CSP models.

7.2 SAT models

The idea for developing a SAT model follows the same principle for a CSP model, except that instead of one dummy variable for a task we introduce a set of dummy Boolean variables. For task T_i, we have $\{V_{i,1}, V_{i,2}, \cdots, V_{i,k}\}$, where k is the minimal number of tasks to be fulfilled. We will have constraints to ensure that task T_i is turned on if and only if one of its associated dummy variables is turned on. To specify turning on one of its respective dummy Boolean variables when T_i is on, we have constraint constraint $(\neg T_i \bigvee_{j=1}^{k} V_{i,j})$, and to specify none of its dummy variable is on when T_i is not on, we have constraints $\bigwedge_{j=1}^{k}(T_i \vee \neg V_{i,j})$. Furthermore, since $\{V_{i,1}, V_{i,2}, \cdots, V_{i,k}\}$ represent T_i, only one of them should be on when T_i is on. In other words, these dummy variables must be mutually exclusive when the respective task is turned on. Therefore, for task T_i, we introduce constraints $\bigwedge_{1 \le u \le k; u \le v \le k}(\neg V_{i,u} \vee \neg V_{i,v})$.

Similarly, to turn on at least k tasks, we specify that one of $\{V_{1,j}, V_{2,j}, \cdots, V_{t,j}\}$ is turned on for $j = 1, 2, \cdots k$, i.e., we have $\bigwedge_{j=1}^{k}(\bigvee_{i=1}^{t} V_{i,j})$.

We call the above additional constraints SAT constraints, which are written as follows.

$$\mathcal{C}_{SAT}^{(k)} = \begin{cases} \bigwedge_{i=1}^{t}(\neg T_i \bigvee_{j=1}^{k} V_{i,j}) \bigwedge_{j=1}^{k}(T_i \vee \neg V_{i,j}) \\ \bigwedge_{i=1}^{t} \bigwedge_{1 \le u \le k; u \le v \le k}(\neg V_{i,u} \vee \neg V_{i,v}) \\ \bigwedge_{j=1}^{k}(\bigvee_{i=1}^{t} V_{i,j}) \end{cases} \tag{8}$$

For our example in Table 1, we have the following SAT constraints for fulfilling two tasks.

$$\mathcal{C}_{SAT}^{(2)} = \begin{cases} (\neg T_1 \vee V_{1,1} \vee V_{1,2}), (\neg T_2 \vee V_{2,1} \vee V_{2,2}), (T_1 \vee \neg V_{1,1}), (T_1 \vee \neg V_{1,2}), (T_2 \vee \neg V_{2,1}) \\ (T_2 \vee \neg V_{2,2}), (\neg V_{1,1} \vee \neg V_{1,2}), (\neg V_{2,1} \vee \neg V_{2,2}), (V_{1,1} \vee V_{2,1}), (V_{1,2} \vee V_{2,2}) \end{cases}$$

Similar to CSP, we can write two SAT models to match the two MAX-SAT models by combining them with $\mathcal{C}_{SAT}^{(k)}$.

8 Experimental Analysis

As shown in Table 3, we now have a total of ten different models in the four categories that are listed in Table 2. These models have different sizes and expressiveness. The objective of this section is to understand the effectiveness of a model and its interaction with a search algorithm.

To this end, we carry out an experimental analysis using systematic and nonsystematic search methods. We chose the existing (and available) algorithms and programs that are either the best or among the best for the models. When no good solver is available to us for a particular model that we need to study, we make our best effort to develop one on our own. The algorithm that will be used on a particular model is also included in Table 3 in parentheses after the model. The first name in the parentheses is for systematic search and the second for nonsystematic search.

variable type	**optimization**	**satisfaction**
general variables	**COP1**/(-, COPsolver)	**CSP1**/(-, NB-WSat)
	COP2/(-, COPsolver)	**CSP2**/(-, NB-WSat)
	COP3/(-, COPsolver)	**CSP3**/(-, NB-WSat)
Boolean variables	**MAX-SAT4**/(MAX-SATsolver, WSat(OIP)	**SAT4**/(SATZ,WalkSat)
	MAX-SAT5/(MAX-SAT solver, WSat(OIP)	**SAT5**/(SATZ,WalkSat)

Table 3. Constraint models and the algorithms to be used on them.

In all our experiments, we set the penalty for an unsatisfied task to be one. This simple penalty function makes a comparison between an optimization approach and a model-checking based approach possible. As discussed earlier, when a decision approach was taken to find the maximal number of tasks to be turned on, a binary search was carried out on a set of CSP or SAT models to find the maximal possible number of tasks that can be satisfied.

We generated random problem instances by changing the probability, called the resource density, that a resource can be used by a task.

8.1 Systematic search

For systematic search, we focused on MAX-SAT and SAT models since no good complete solvers for COP and CSP were available to us. For the SAT models, we adopted SATZ [6], one of the best complete SAT solvers based on Davis-Putman-Longmann-Loveland (DPLL) algorithm [2]. We developed an algorithm for MAX-SAT models, which directly extends the Davis-Putman-Longmann-Loveland (DPLL) algorithm [2] and we call it MAX-SATsolver.

In our experiments, we used random problem instances with various sizes, ranging from five to ten tasks. Every task has three resource requirements, so that the complexity of a problem instance is NP-hard in the worst case, as proved in Section 3. We chose these small problem sizes in order to solve the problems to optimality using the systematic search algorithms. The resource densities of the random problem instances increase from 0.1 to 0.9 with an increment of 0.1. For a pair of problem size and resource density, we generated 100 instances and averaged the results over the 100 trials. The results on six tasks and three requirements per task are shown in Figures 1 and 2. As the vertical axes in the figures are in a logarithmic scale, the complexity of the algorithms grows exponentially with the number of resources and the resource density. One conclusion from Figure 1 is that model MAX-SAT4 is more effective than model MAX-SAT5. For example, for problems with ten resources and 0.4 resource density, an optimal solution can be found on MAX-SAT4 in 425 seconds on average, while it needs more than 1700 seconds on MAX-SAT5.

We compared the two SAT models on the same set of problem instances using SATZ. Figure 2 shows the result. In contrast to the results on MAX-SAT models, SAT5 is more effective than SAT4. One plausible explanation for model SAT5 being more effective than model SAT4 is the following. Recall that models SAT4 and SAT5 are direct extensions of models MAX-SAT4 and MAX-SAT5, in which it is specified to turn on at

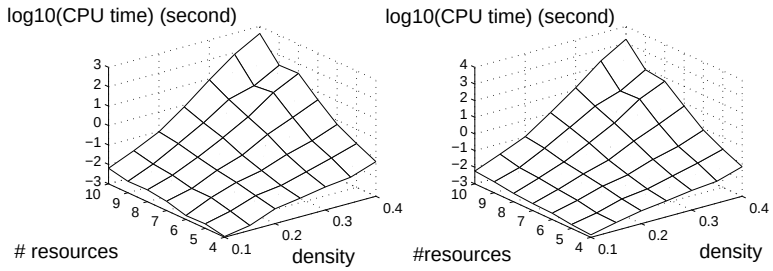

Fig. 1. CPU time of MAX-SATsolver, in a logarithmic scale, on models MAX-SAT4 (left) and MAX-SAT5 (right) (6 tasks and 3 requirements/task).

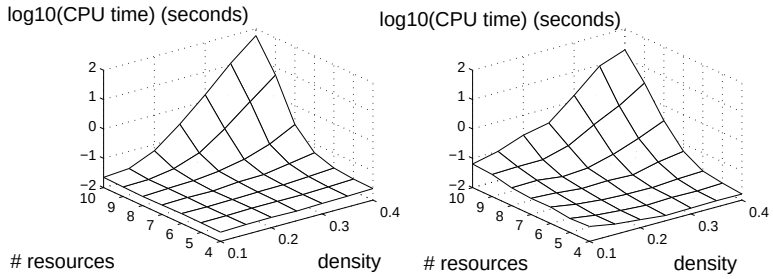

Fig. 2. CPU time of SATZ, in a logarithmic scale, on models SAT4 (left) and SAT5 (right) (6 tasks and 3 requirements/task).

least a certain number of tasks. Recall that MAX-SAT5 is more expressive than MAX-SAT4 by bringing some hidden constraints to bear. By the same token, SAT5 is more expressive than SAT4 with more constraints introduced. Therefore, the ratio of number of clauses versus the number of variables is higher for SAT5 than that for SAT4 for a given target number of tasks to be satisfied. When searching for the maximal number of tasks to be turned on, many unsatisfiable SAT models will be checked. These unsatisfiable models are substantially larger than satisfiable models for the same problem, as the former has to encode the fact that more tasks are to be fulfilled. As a result, a search algorithm typically spends more time on an unsatisfiable model than a satisfiable one. Furthermore, deciding if an unsatisfiable SAT model is unsatisfiable is relatively easier when there are more constraints and the number of variables is fixed. This observation is in line with the easy-hard-easy phase transition phenomenon of 3-SAT [8]. Combining these facts, therefore, we can speculate that model SAT5 must be more effective than SAT4.

In addition, while the complexity of SATZ still grows exponentially with the number of resources and resource density, it is smaller than that of our MAX-SATsolver on MAX-SAT models. For example, for problems with ten resources and 0.4 resource density, SATZ finishes in 59 seconds on average while the MAX-SAT algorithm needs 425 seconds. Although the results in Figures 1 and 2 indicate that a combination of a SAT model and SATZ algorithm is better than a combination of a MAX-SAT model

and our MAX-SATsolver, we need to mention two issues related to this comparison. First, our MAX-SAT algorithm and implementation are not optimized and they can be further improved. For example, many lookahead techniques for SAT algorithms may be exploited to speed up the MAX-SATsolver. Second, we only considered the problem of turning on the maximal number of tasks. More importantly, when tasks have different penalties and the goal is to minimize the total penalty of all unfulfilled tasks, we can only rely on the MAX-SAT Models and a MAX-SAT algorithm.

8.2 Local search

We considered all the models using nonsystematic search methods. We used WalkSat [11, 10], the most celebrated local search algorithm for SAT problems, for our SAT models. We ran NB-WSat [4] on our CSP models and chose WSat(OIP) [13] for the MAX-SAT models. NB-WSat and WSat(OIP) are direct extensions of WalkSat, but along different directions. NB-WSat extends WalkSat to include non-Boolean variables. WSat(OIP) expands WalkSat to handle constraint optimization problems. We leave the details of these algorithms to their original description in [11, 10, 4, 13] These algorithms are perhaps the best for the models for which they will be applied. Finally, we developed our own local search algorithm for COP models, which we call COPsolver. This solver is also an extension of WalkSat to deal with non-Boolean variables and constraint optimization problems at the same time. One significant deviation of this COPsolver from WalkSat is that an unsatisfied constraint of the highest weight among all unsatisfied ones is selected in each step. When more than one such highest-weight constraints exist, the tie is broken randomly. The detail of this constraint solver will be reported elsewhere.

Again we used random problem instances with three resource requirements per task to make the problem instances difficult. We tried some problem instances in the range with the number of tasks going from 30 to 100, the number of resources ranging from one to two times of the number of tasks, and the density of resource requirements spanning from 0.1 to 0.9 for problems with 30, 40 and 50 tasks, and 0.1 to 0.5 for the larger problem sizes.

We run WalkSat version 39 with nine random restarts (giving a total of 10 runs) on each problem instance, and set each run to 100,000 flips. The noise ratio was set to 50/100. We used NB-WSat version 4 and adopted the same set of parameters as for WalkSat. We only applied WSat(OIP) once to each problem instance, and let it run 50,000 moves. We set its noise probability at 0.01. For our own COPsolver, we gave a total of 600 seconds or 10 minutes of CPU time. If the algorithm makes more than 5,000 plateau moves, it will restart from a random starting point.

For performance of the algorithms on a specific instance, we take into account the quality of the best solution they provided and the time in which the best solution was found (not the total CPU time on the instance).

The results on CPU time on the instances of 60 tasks and 80 resources are given in Table 4, averaged over 10 instances. On these problem instances, all algorithms produced the same solution quality of fulfilling 26 tasks on average. The test results on other problem sizes exhibit similar trends.

Based on Table 4, we can make a few observations on the models and the search algorithms. Going from the models in the COP (MAX-SAT) family to those in the CSP

p	COPsolver			NB-WSat			WSat(OIP)		WalkSat	
	COP1	**COP2**	**COP3**	**CSP1**	**CSP2**	**CSP3**	**MSAT4**	**MSAT5**	**SAT4**	**SAT5**
0.1	5.46	0.09	0.92	168.67	49.62	100.06	0.20	0.80	41.94	51.22
0.2	22.68	0.03	4.04	494.19	81.41	287.63	1.60	2.00	80.24	108.49
0.3	53.63	0.07	19.91	1186.67	132.26	551.56	3.60	7.00	131.29	184.68
0.4	106.90	0.12	86.73	2076.74	191.29	1759.58	7.80	12.40	178.52	257.19
0.5	498.21	0.22	132.45	10198.70	269.28	9599.19	16.00	29.60	235.41	339.65

Table 4. Average CPU time (in seconds) when the best solutions were found on instances of 60 tasks, 3 resource requirements per task, and 80 resources.

(SAT) family, the relative effectiveness of different models within the same category seems to be preserved from one algorithm to another algorithm. Specifically, the order of strengths within COP family is COP2, followed by COP3 and then by COP1; a similar order appears in the CSP family, i.e., CSP2, CSP3 and then CSP1. The same observation can be made across the MAX-SAT and SAT families. These observations can lead us to some interesting conclusions. First, within COP and CSP families, the models using resources as variables (i.e., COP2, COP3, CSP2 and CSP3) are more effective than the models using resource requirements as variables (i.e., COP1 and CSP1). Second, making a model more expressive does not pay off for local search. COP3 and CSP3 are more explicit models than COP2 and CSP2. However, COPsolver and NB-WSat run longer on the more expressive models. Similar observation can be made between MAX-SAT4 (SAT4) and MAX-SAT5 (SAT5). This result for local search contradicts with that of SATZ on SAT4 and SAT5. Obviously, the expressiveness has different impact on different search methods.

Now consider the best combinations of algorithms and models from all categories. The COPsolver/COP2 pair is the champion of all. Its CPU time is typically more than an order of magnitudes smaller than the second best pair, WSat(OIP)/MAX-SAT4. NB-WSat/CSP2 is the slowest. The sizes of actual CSP models that NB-WSat runs on, in terms of megabyes of memory, are usually significantly larger than the equivalent models used by other solvers. This is partly due to the fact that NB-WSat introduces a Boolean variable for each variable value. It remains to be clarified if NB-WSat's performance can be improved if the memory requirement can be reduced using some engineering effort.

The results in Table 4 clearly show that the combinations of decision models and CSP or SAT solvers cannot compete with the combinations of optimization models and COP or MAX-SAT solvers. This indicates that the approach of model checking is not appropriate for the resource allocation problem considered here. The situation will become worse if tasks have different penalties if unfulfilled.

Since COPsolver and WSat(OIP) perform significantly better than the two decision-based solvers, we compared them on larger problems. Table 5 shows their CPU times on problems of 100 tasks and 200 resources, averaged over 10 trials. Both algorithms can fulfill the same number of 66 tasks. However, COPsolver is able to find the solutions a few orders of magnitudes sooner than its counterpart. To further our understanding,

p	COPsolver	WSat(OIP)
0.1	0.71	13.21
0.2	0.92	86.67
0.3	1.65	276.33
0.4	3.32	601.33
0.5	6.09	1175.10

Table 5. Average CPU time of WSat(OIP) and COPsolver (in seconds) when the best solutions were found on instances of 100 tasks, 3 resource requirements per task, and 200 resources.

	COPsolver		WSat(OIP)	
p	tasks	time	tasks	time
0.1	55.1	3.0	66.6	14.4
0.2	56.1	12.0	65.2	74.1
0.3	52.7	19.5	63.1	220.4
0.4	54.6	65.1	62.4	497.1
0.5	53.7	93.0	62.1	1526.0

Table 6. Average solution quality (in the number of tasks turned on) and average CPU time (in seconds) of WSat(OIP) and COPsolver when the best solutions were found. Problem instances have 100 tasks, 200 resources and a random number of resource requirements per task which is uniformly chosen from 1 to 10.

we used problem instances with variable number of resource requirements per task within a problem. The average results of 10 problem instances are in Table 6. The tasks in a problem may have a different numbers of resource requirements, uniformly ranging from 1 to 10. As the results indicate, WSat(OIP) is more effective, finding better solutions than COPsolver, while uses more time to do so. In our experiments, we also increased the allowed CPU time for the COPsolver. However, it did not produce better solutions. This implies that the local search space of COPsolver is restricted. Some of the real problem instances provided by the authors of [3] also have variable requirements, and the relative strengthes of COPsolver and WSat(OIP) exhibit a similar pattern on these problem instances.

These experimental results indicate that the features of this resource allocation problem indeed interplay with search algorithms. It also means that although it is more effective and efficient than the CSP and SAT solvers, COPsolver can still be improved and future research is in order. Furthermore, WSat(OIP) can be improved as well. As pointed out in [13], it remains to be investigated if WSat(OIP) is suitable for problems with intricate solution structures. Our resource allocation problem indeed has its own tight structures, which may cripple WSat(OIP) and make it not efficient.

9 Related Work and Discussions

The closest work is [3]. In fact, the resource allocation problem considered in this paper was directly brought to our attention by the authors of that paper. The constraint models

we developed were inspired by the modeling work done there. Our MAX-SAT Model 5 is in fact the model studied in [3]. Comparing to the work in [3], we have made several contributions. First, we introduce soft constraint techniques to address the problem and develop a family of ten different models suitable for different optimization criteria. Our models can support both satisfaction and optimization problem-solving strategies. Second, we prove that the worst-case complexity of the problem is NP-complete. This result strongly supports the current practice of using local search techniques for finding high-quality approximation solutions to large problem instances [3].

There is a large body of research on modeling using CSP and soft CSP. Along this line, our work can be considered as a direct application of soft constraint approaches [1, 9] for modeling complex constraint problems.

In essence, the method of using decision or CSP models follows the spirit of model checking [7]. Model checking has been shown very effective and efficient on many formal verification problems and planning problems, and the amount of publications on these topics is overwhelming. However, on the resource allocation problem studied here, model checking is inferior to directly solving using optimization methods. This is perhaps mainly due to the difficulty of developing the right decision models at the first place.

10 Conclusions

We have formulated a resource allocation problem, which is a central part of a real-world crew scheduling problem, as a soft constraint satisfaction and optimization problem and proved that the problem is NP-complete. Using this resource allocation problem as a case study, we investigated modeling choices in developing soft constraint models. We proposed four types of models for the problem and ten soft constraint models with different optimization objectives and expressiveness. We then applied the best known systematic and nonsystematic search algorithms to solve the resource allocation problem and to analyze the interplay of modeling choice and search algorithm choice. Among others, we can draw two important conclusions from our experimental study. First, soft constraint techniques are very effective for complex constraint problems, such as the one considered in this paper. Effective constraint modeling can give rise to constraint models that can be efficiently solved. Second, decision models and a model checking approach are not suitable for the inherited optimization problem embedded in the resource allocation problem. In all the problem instances we have tested in our analysis using local search, solving the optimization problem directly is more efficient than solving a set of decision problems.

References

1. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, 1997.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of ACM*, 5:394–397, 1962.
3. M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches. Marbles: A family of cooperative negotiation schemes for real-time fault-tolerant distributed resource allocation.

In *Proc. AAAI-01 Fall Symp. on Negotiation Methods for Autonomous Cooperative Systems*, 2001.
4. A. M. Frisch and T. J. Peugniez. Solving non-Boolean satisfiability problems with stochastic local search. In *Proc. IJCAI-01*, pages 282–288, 2001.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 1979.
6. C. M. LI and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. IJCAI-97*, pages 366–371.
7. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
8. D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. AAAI-92*, pages 459–465, 1992.
9. T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: Hard and easy problems. In *Proc. IJCAI-95*, pages 631–637.
10. B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. AAAI-94*.
11. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proc. AAAI-92*, pages 440–446.
12. B. M. Smith. Modelling a permutation problem. In *Proc. of ECAI-00 Workshop on Modelling and Solving Problems with Constraints*, 2000.
13. J. P. Walser. *Integer Optimization by Local Search*. Springer, 1999.
14. T. Walsh. Permutation problems and channeling constraints. In *Proc. IJCAI-01 Workshop on Modeling and Solving Problems with Constraints*, pages 125–133.