

Modularity and Complexity Profiles in Overconstrained Resource Allocation Problems

Alejandro Bugacov[†], Donghan Kim[†], Carla Gomes[‡] and Bart Selman[‡]

([†]) Information Sciences Institute, University of Southern California
4676 Admiralty Way Marina del Rey, California 90292

([‡]) Department of Computer Science, Cornell University, Ithaca, NY 14853

(bugacov,donghank)@isi.edu, (selman,gomes)@cs.cornell.edu

Abstract

We propose a technique for solving a type of over-constrained resource allocation problem that is commonly encountered at the heart of many real-world planning and scheduling applications. This type of resource allocation problems are characterized by what we call modular constraints: good solutions are those that guarantee the allocation of resources to tasks that can satisfy all of its requirements and avoid generating assignments that result in broken or partially satisfied tasks. Using SAT and Pseudo-Boolean encodings, we show that a typical or plain encoding results in solutions with a large number of broken tasks while our proposed encodings, by taking into account the modular constraint structure of the problems, guarantee good solutions. In addition, we show that the phase-transition characteristics of the resulting SAT formulas can be used to rapidly estimate the maximum number of satisfiable tasks. This information can be very valuable in the design and implementation of transition-aware solvers.

Introduction

Recent advances in constraint-based reasoning have led to substantial improvements in planning and scheduling systems. For example, the Graphplan (Blum and Furst 1997) and Blackbox (Kautz and Selman 1999) planners have significantly extended the range of feasible planning tasks (Weld 1999). A common feature of these approaches is that the planning task is translated into a set of constraints, either as a Boolean satisfiability (SAT) encoding or a more general constraint satisfaction formulation. Instead of search techniques specifically tailored to a particular planning task, general SAT solvers or CSP engines are used to find a satisfying solution (if one exists) for the set of constraints. (In fact, the solvers have no notion of goal state, initial state, or operators.) The general solvers are competitive with, and often outperform, specialized search methods on a range of planning benchmarks.

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

In this paper, we extend the constraint-based approach to the case where we cannot fulfill all desired objectives of the goal state due to resource constraints. To be more specific, we consider a challenging real-world resource allocation problem. Our resource allocation problem is inspired by the complex issue of resolving resource conflicts in the scheduling of flight training exercises. In this domain, beside the satisfaction of a series of constraints, the scheduling of a training exercise (or task) depends on the availability of qualified resources for every requirement of the task and will fail to get scheduled otherwise. At the same time, the scheduling of some tasks is more valuable than others and thus there is also an optimization component to the problem. Frank et al. (Frank et al. 2001) introduced a model resource allocation problem capturing these features together with some distributed solving schemes that they refer to as Marbles. The Marbles problem plays a fundamental role in the efficient allocation of resources to tasks in the CAMERA/SNAP negotiations-based flight scheduling system developed at USC ISI (for more information see www.isi.edu/camera). Recently, it has been shown (Zhang 2001) that finding the optimal solution to this problem is NP-hard. We focus on the development of Boolean (and Pseudo-Boolean) modular encodings of this problem that result in proper solutions and contain valuable information about the complexity characteristics of the problems that can lead to the implementation of phase-transition aware solvers for this domain.

It is fairly straightforward to capture the underlying constraints in a Boolean SAT encoding (or, if we want to include numerical information, in the somewhat richer form of a pseudo-Boolean representation (Walser 1997)). In a typical or "plain" SAT encoding of constraint satisfaction problems of this kind, one will begin by associating Boolean variables to the allocation of a resource to a given task and then create a Boolean formula in conjunctive normal form (CNF) consisting of two main types of clauses: 1) those stating that at least one resource must be allocated to a given

task (or requirement in a task) and 2) those precluding a resource to be allocated to more than one task at a time. Obtaining a solution to the problem will then amount to finding a truth assignment of the variables that will satisfy the whole formula. Unfortunately, such a solution will only be likely to exist if the problem is so under-constrained that there are enough resources to satisfy all tasks: a case that is of no practical importance to most real-world applications which usually run over-constrained or at the limit of their capacity. Since the problem is over-constrained, we could use a MAXSAT approach or other optimization technique to find a schedule that satisfies the largest possible number of constraints. Unfortunately, because of the uniform nature of the constraint representation, the resulting schedules will contain a large number of *broken tasks* for which some of the required resources are not available. The problem we are encountering is that the constraints are represented in a uniform manner, whereas in the underlying problem domain, we are really dealing with a modular constraint structure, where each task corresponds to a specific set of constraints. Only when we schedule a task, do we want to "activate" the constraints corresponding to that task.

We therefore explore instead a special encoding of our problem domain. In this so-called modular encoding, we introduce a special set of Boolean variables, called task activation variables. We modify our encoding in such a way that the sets of constraints for a certain task are only activated when the corresponding task activation variable is set to True. To obtain a pure decision problem, we also introduce clauses that state that at least K of the task selection variables should be activated. Varying the value of K , either by a binary search or incrementally from an initial guess, we can rapidly find the maximum number of tasks that can be filled with the current resource level and imposed constraints. In our pseudo-Boolean encoding, we do not need to add the clauses that select at least K tasks, rather we use a set of soft weighted constraints to directly optimized the number of selected (weighted) tasks.

Both our modular SAT and pseudo-Boolean encodings allow again for the use of state-of-the-art solvers to find schedules. The key question is whether our special modular encoding preserves the advantages of a constraint-based approach. In particular, it could be the case that the encodings become much harder to solve. We present a detailed empirical evaluation of our approach and show how the modular encodings can still be solved effectively. We also show how our approach outperforms a special purpose scheduling procedure tailored directly towards the domain.

		Resources								
Task[value]		A	B	C	D	E	F	G	H	I
T_1 [300]	Q_1	•	•							
	Q_2				•		•			
	Q_3								•	•
T_2 [600]	Q_1	•		•						
	Q_2					•	•			
	Q_3							•	•	
T_3 [200]	Q_1		•	•						
	Q_2						•			
	Q_3									•

Figure 1: Example of a Marbles resource allocation problem with 3 tasks and 9 resources

Marbles Resource Allocation Problem

The resource allocation problem that we consider for our encodings can be formulated as follows. There is a collection \mathbf{R} of M available resources and a set of N tasks $T = T_1, T_2, \dots, T_N$. Each task T_i has a domain value V_i and a set of q_i requirements. Each requirement Q_j of task T_i has a different list \mathbf{P}_{ij} of possible or qualified resources, with $\mathbf{P}_{ij} \in \mathbf{R}$. A task needs to acquire one resource for each of its requirements and have all its requirements filled in order to be executed and add up its domain value to the total value of a solution. (This is the condition that imposes the modular constraints.) Figure 1 shows an example problem with four tasks and nine resources. In the example each task has three requirements but in the general case the number of requirements can vary from task to task. A • mark in a given cell indicates that the resource of the corresponding column is qualified or is a possible candidate for the requirement of the corresponding row. If a task partially fills its requirements (broken task) it does not get scheduled and it does not contribute to the total domain value of the solution.

In a flight training scheduling application, each task will represent a flight training exercise and the requirements represent different types of resources that the task needs to obtain in order to get scheduled (e.g., 1 Pilot, 1 Airplane and 1 Range). In addition, there is a value associated to each training exercise representing some domain value (e.g., training readiness or flight hours) that a task will contribute to the squadron total value if it gets scheduled. In the following sections we describe various techniques to encode this problem into a set of constraints that enable the use of state-of-the-art off-the-shelf solver.

SAT(K) Modular Encoding

In our approach, we focus on the formulation of a SAT encoding of the Marbles problem described above (see Figure 1) that will guarantee resource assignments sat-

isfying at least K of the N tasks forming the problem. For each value of K ($K \leq N$), the encoding generates a CNF formula, that if solvable, guarantees that from the original set of N tasks at least K are filled. (A task is filled when it finds one available resource for each of its requirements.) For a given value of K , we start our formulation by composing a SAT formula of the following form

$$f = (f_K) \wedge (f_{cross}) \wedge (f_1 \wedge f_2 \wedge f_3 \wedge \dots \wedge f_N) \quad (1)$$

The formula f represents the problem of finding resource assignments for at least K tasks and is in turn the conjunction of three main sub-formulas f_K , f_{cross} and f_i ($i = 1, \dots, N$). The formula f_K contains the clauses responsible for selecting at least K different tasks, f_{cross} is the formula containing resource variables across tasks and requirements and is formed by clauses restricting a given resource from being assigned to more than one requirement at a time and f_i ($i = 1, 2, \dots, N$) contains the clauses responsible for selecting at least one, and optionally no more than one, resource for each requirement in task i .

The f_K formula is completely independent of the resources and is the only term in Eq. 1 that depends on the value of K . The clauses in f_K are formed by two types of Boolean variables: task activation variables and dummy variables. For a problem with N tasks we introduce N Boolean task activation variables $m_1, m_2, m_3, \dots, m_N$, where each variable m_i represents a task T_i and its activation (i.e., $m_i = \text{TRUE}$) indicates that T_i got filled in a given solution. In addition, we define $K \times N$ dummy variables:

$$a_1, a_2, \dots, a_N ; b_1, b_2, \dots, b_N ; k_1, k_2, \dots, k_N \quad (2)$$

to select K different task activation variables such that in a given solution at least K of the N activation variables will be set to TRUE. These dummy variables are introduced with the conditions that $x_i \rightarrow \bar{y}_i$ for any pair of variables and they are related to the tasks activation variables by

$$a_i \rightarrow m_i, b_i \rightarrow m_i, c_i \rightarrow m_i, \dots, k_i \rightarrow m_i \quad (3)$$

for $i = 1, 2, \dots, N$.

The formula f_{cross} is introduced to preclude resources from being assigned to more than one requirement at a time. At this point we introduce one Boolean resource variable X_{ij} for each \bullet marked cell in the grid (shown in Figure 1) and where $X_{ij} = \text{TRUE}$ indicates that resource X is assigned to task i requirement j . With this notation, f_{cross} contains all possible binary clauses of the form $\bar{X}_{ij} \vee \bar{X}_{kl}$, with $X \in R$ and all possible pairs of variables with $j \neq l$ if $i = k$ and $i \neq k$ if $j = l$.

For each individual task T_i we define a formula f_i formed by two types of clauses encoding the following

two conditions for each requirement of the task: 1) at least one of its possible resources must be selected and, 2) only one resource per requirement should be selected. In order to avoid having broken or partially filled tasks, we introduce in each of these clauses the corresponding task activation variable negated, such that the clauses in f_i are only active when $m_i = \text{TRUE}$ and trivially satisfied otherwise.

It is worth noting that thanks to the mechanism adopted in f_K to select K different task activation variables, the resulting total number of clauses and variables in the SAT(K) encoding scales polynomially with the size of the problem as $O(N^3)$.

Pseudo-Boolean Modular Encoding

In order to obtain a more compact encoding of the problem, we extended the modular encoding idea used in SAT(K) to an integer programming encoding using Pseudo-Boolean (PB) variables (Walser 1997). In this encoding variables can only take values 0 or 1 and all constraints are formulated as inequalities. Thus, the large number of SAT clauses introduced in SAT(K) to exclude resource contention across requirements can be combined in a single constraint and therefore the number of resulting constraints is substantially lower than the number of clauses in the SAT(K) encoding. At the same time, we can introduce weights and differentiate between soft and hard constraints, what makes this approach very attractive for finding optimal solutions to the problem.

In the PB encoding we define the same set of resource and tasks activation variables as introduced in the SAT(K) encoding and decompose the problem in a similar fashion (see Eq. 1), except for the fact that the constraints corresponding to f_K for selecting at least K of the N variables are substituted by a set of N weighted soft constraints. The set of exclusive clauses corresponding to f_{cross} are now written as the following set of M integer inequalities (one for each resource $X \in R$):

$$\sum_{i=1}^N \sum_{j=1}^{r_i} \bar{X}_{ij} \geq \sum_{i=1}^N r_i - 1 \quad (4)$$

where r_i is the number of requirements in task i for which X is a possible resource, $X_{ij} = 0$ or 1 and a bar over a variable denotes its negated value. In a similar fashion, the clauses in each of the f_i formulas expressing the constraints for the selection of resources within each requirement j of task i , are now written as:

$$\sum_{l=1}^N R_l + \bar{m}_i \geq 1 \quad \text{with } R_l \in \mathbf{P}_{ij} \quad (5)$$

where \mathbf{P}_{ij} is the set of possible resources for requirement j of task i and m_i is the task activation variable

representing task i which is evaluated by a soft constraint *soft* : $V_i m_i \geq 0$ weighted by V_i , the domain value associated to task i .

We seek a solution to the system of integer constraints that satisfies all hard constraints while minimizing the number of violated soft constraints.

Simulated Annealing Solution

With the purpose of testing the efficacy of our SAT encodings we implemented a simple stochastic solver based on the concept of simulated annealing (SA) (Kirkpatrick, Gelatt and Vecchi 1983). In this solver, first we generate a random initial assignment or state by allocating qualified resources to tasks until all resources have been assigned (the case where all tasks are satisfied before we run out of resources is not of much interest in this work) and then we start a stochastic hill-climbing process by swapping resource assignments among different requirements and tasks. The algorithm does *maxFlips* iterations of this kind and at each iteration step we evaluate the resulting total domain value corresponding to the new assignment, and accept the move with probability 1 if the new value is larger or equal than the previous one and with probability $\exp\left(\frac{-(oldValue - newValue)}{T}\right)$ otherwise, where T is the temperature parameter that takes the values prescribed by a selected annealing schedule.

Critical Number of Scheduable Tasks Prediction

The phase-transition phenomena observed in 3-SAT and other combinatorial problems has been studied in great detail in the past 10 years (Cheeseman, Kanefsky and Taylor 1991; Kirkpatrick and Selman 1994). A key envisioned application of such phenomena in real-world problems is the development of transition-aware solvers that can reason about their proximity to the critical region and make real-time smart trade-offs between solution quality and computational time. A main practical issue is to identify the order parameter of the transition diagram and the corresponding variables in the application domain. Using our SAT(K) encoding of the Marbles resource allocation problem, we have empirically found that the parameter γ given by

$$\gamma = \frac{\text{Number of Active Clauses}}{(\text{Number of Resource Variables})^{1.48}} \quad (6)$$

can be identified as a reasonable order parameter. For a problem with N tasks and M different resources and where each task has, in average, r requirements and each resource is a possible resource for a requirement with probability p , the average number of resource variables can be approximated by $rpNM$. On a given

formula resulting from the SAT(K) encoding, the number of active clauses is given by the number of clauses that are not trivially satisfied by the presence of a task activation variable and can be expressed as a quadratic formula in K . Therefore, from the critical value of γ , we can derive an estimate of the critical value of K by solving the following quadratic equation

$$\begin{aligned} a_2 K_c^2 + a_1 K_c + (a_0 - \gamma_c) &= 0 & (7) \\ a_2 &= \frac{(1 + N/2)}{(rpNM)^{1.48}} \\ a_1 &= \frac{[2 - N/2 + r(1 + pM(pM - 1)/2)]}{(rpMN)^{1.48}} \\ a_0 &= (rpN - 1)/2 \end{aligned}$$

In the next section we present experimental evidence that this prediction mechanism can provide very reasonable estimates of the maximum number of tasks that a solver might schedule before reaching the critical region where a sharp exponential increase in the computational time is expected in order to satisfy additional tasks.

Experiments and Results

We tested the performance and efficacy of our encoding techniques in a large number of resource allocation problems with the same characteristics of the one described in Figure 1. These problems represent typical situations found in the resource allocation part of the scheduling process of the CAMERA/SNAP system. CAMERA/SNAP is a negotiations-based system developed at USC ISI which is currently being used for scheduling of real-world flight training exercises. Using the SAT(K) and Pseudo-Boolean (PB) encodings and simulated annealing (SA) approaches described above we solved problems with the number of resources, M , and the number of tasks ranging from 30 to 190. The SAT and PB encodings were solved using the state-of-the-art local search engines for this problems: walkSAT (walksat v.35 available from www.satlib.org) and WSAT(OIP) (www.ps.uni-sb.de/walser/), respectively.

Figure 2 clearly supports the claim that a modular approach is needed to properly encode the problem. In the figures we plot the number of filled and broken tasks ((a) and (b), respectively) as a function of the resource deficit, i.e., the average number of requirements per task times the number of tasks minus the number of available resources. Figure 2 compares results obtained using the modular SAT(K) approach to those given by a plain SAT encoding for problems with 30 tasks and the number of resources varying from 30 to 100. In Figure 2a, we see that for negative values of the

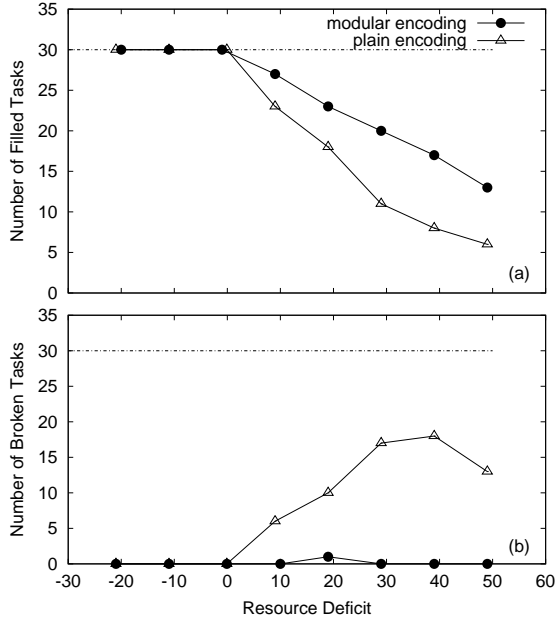


Figure 2: Evidence that a modular approach is needed to properly encode the presented problems. The figure shows the number of filled tasks (a) and the number of broken tasks (b) as a function of the resource deficit when a modular or a plain encoding of the problem is used.

resource deficit the problem is under-constrained and both encodings manage to fill all tasks. As the resource deficit increases the problem becomes over-constrained and the gap between the two curves grows significantly to a point where the modular encoding fills almost 3 times more tasks than the plain encoding. Looking at Figure 2b we see that in the plain encoding case the large fraction of unfilled tasks is a result of the inefficient assignment of resources to broken tasks while in the modular encoding the number of broken tasks is zero in most cases and the fraction of unfilled task is formed by empty tasks that cannot be satisfied due to resource insufficiency.

In Figure 3a and b we compare the efficiency of solutions for three of the approaches described above: SAT(K), PB and SA. Figure 3a shows the computational time as a function of K , the least number of filled tasks in the SAT(K) approach. Full circles correspond to the SAT(K) values while the dashed and dotted lines show the values obtained with PB and SA, respectively. (These last two approaches are independent of K ; thus, we have marked their values as horizontal lines.) We see that for most values of N , SAT(K) tends to be the fastest solver for low values of K but it becomes about an order of magnitude slower than PB when K reaches the critical region. As expected, both SAT(K) and PB outperform the SA approach. In Figure 3b we plot the

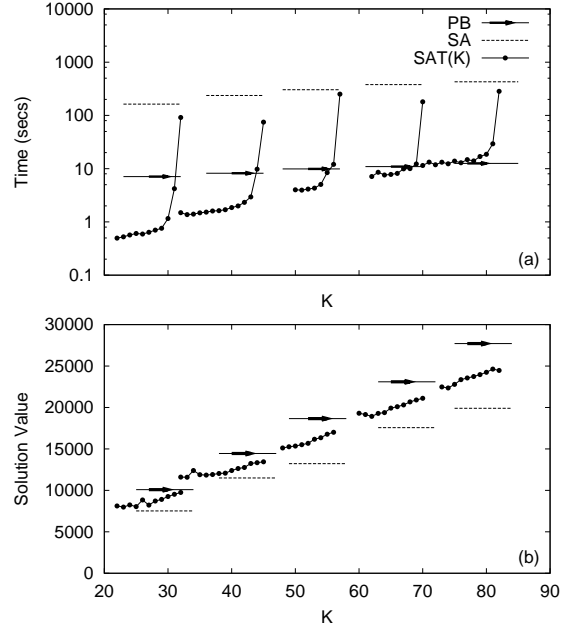


Figure 3: (a) Computational time and (b) Solution value as a function of the least number of filled tasks, K , for problems with N tasks and N resources.

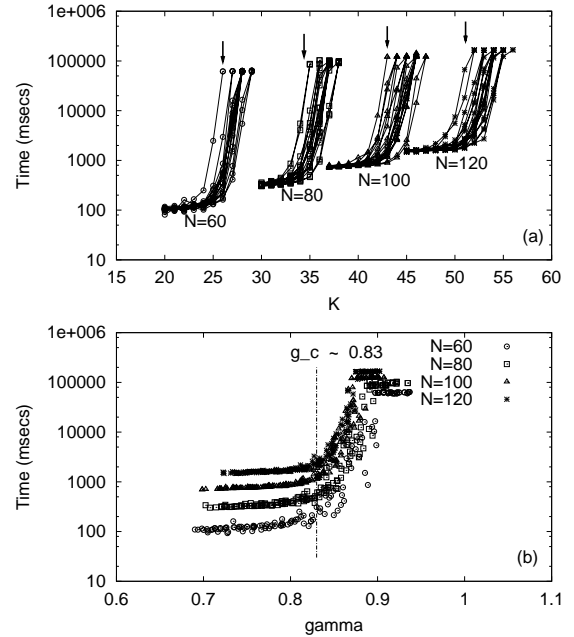


Figure 4: (a) Phase-Transition profiles of the SAT(K) encoding for a large collection of problems with N tasks and N resources. The vertical arrows indicate the predicted values of K_c . (b) Reparametrization of the curves in (a) as a function of γ , an empirical ratio of the number of active clauses to a power of the number of resource variables.

solution value as a function of K . (The solution value is the sum of the domain values of the satisfied tasks.) We see that in order to obtain solutions of comparable value $SAT(K)$ takes much longer than PB in even so cannot reach a solution close to the PB value. This result is a combination of two issues. First, in contrast to PB, where we introduce weighted soft constraints and in SA where we hill-climb using the solution value, the $SAT(K)$ approach is not really tuned to the optimization of the solution value. Second, since PB results in a more compact encoding, the size of the problem that the WSAT(IOP) solver is actually solving is significantly smaller than the one being solved by walkSAT.

In Figure 4a we show the computational time needed to find a solution of the $SAT(K)$ encoding with at least K satisfied tasks for several problems with N tasks and N resources with N ranging from 60 to 120. We can clearly see that all problems display a similar easy-hard phase-transition behavior and where the computational cost encounters a sharp *elbow* as K approaches the maximum number of satisfiable tasks and the problems become overconstrained. In order to use of this information in the implementation of transition-aware solvers, one would need to find a direct correlation between the position of the elbows and the characteristics of the corresponding boolean formulas. Figure 4b, shows how the empirical parameter γ , introduced in Eq. 6 and given by the ratio of the number of active clauses to a power of the number of resource variables, provides a very reasonable scaling of the position of the elbows purely in terms of the characteristics of the formulas. Choosing a value of $\gamma_c \approx 0.83$ and solving for the critical value of K given by Eq. 7 we find the values of K_c shown by the 4 vertical arrows in Figure 4a, which provide a very reasonable prediction of the maximum number of tasks that a solver might be able to satisfy before hitting the hard region of the corresponding phase-transition curve. The fact that this prediction mechanism is purely based on counting the number of clauses and variables in a corresponding boolean formula, suggests that this information can be very quickly transmitted to a generic solver to help it make smart decisions in real-time.

Conclusions

Recent developments in constraint-based approaches combined with very fast solvers make it appealing to use such techniques for tackling planning and scheduling problems. We extend the constraint-based planning and scheduling approach using an encoding technique that is very well suited for solving over-constrained resource allocation problems with modular constraints. Such type of resource allocation problems are commonly found in real-world planning and scheduling applications, like the scheduling of flight training missions.

Our modular encoding is very suitable to capture the complementarity of constraints, and our results clearly demonstrate the potential of our approach.

We presented a special encoding technique for our problem domain, the so-called modular encoding, and through detailed empirical evaluation of this approach we showed that the modular encoding properly captures the nature of the problem and minimized the creation of partially filled tasks. We use this approach both in a SAT and a Pseudo-Boolean encoding and find that overall, due to the compactness of the representation, both in time and solution quality, the Pseudo-Boolean encoding outperforms SAT. Studying the phase-transition characteristics of the SAT formulas coming from the modular encoding of large collections of problems in our domain, we empirically derive a phase-transition based prediction mechanism that can be used to rapidly estimate the maximum number of tasks that a solver might be able to satisfy before reaching the hard region of the easy-hard phase-transition curve. This information can be extremely very valuable in the design and implementation of transition-aware solvers.

Acknowledgments

We gratefully acknowledge funding by DARPA ITO ANTS program (Contract No. F30602-00-2-0533). Alejandro Bugacov thanks the members of ISI/CAMERA and Andrew Parkes for very fruitful discussions and support.

References

- Blum, A, and Furst, M; Fast Planning Through Planning Graph Analysis, *Artificial Intelligence*, 90:281–300 (1997).
- Cheeseman, P.; Kanefsky, R.; and Taylor, W.; Where the Really Hard Problems Are; *Proc. IJCAI-91*, 163-169, (1991).
- Frank, M; Bugacov, A; Chen, J; Dakin, G; Szekely, P; Neches, R.T; The Marbles Manifesto: A Definition and Comparison of Cooperative Negotiation Schemes for Distributed Resource Allocation; *Proceedings of AAAI Fall Symposium 2001*, Cape Cod, MA (2001).
- Kautz, H, and Selman, B; Unifying SAT-based and Graph-based Planning. *Proc. IJCAI-99*, (1999).
- Kirkpatrick, S.; Gelatt, C.; and Vecchi, M; Optimization by Simulated Annealing. *Science*, 220, 671-680, (1983).
- Kirkpatrick, S. and Selman, B; Critical behavior in the satisfiability of Boolean expressions. *Science*, 264, (1994).
- Selman, B.; Kautz, H.; and Cohen, B. Noise Strategies for Improving Local Search. *Proceedings of AAAI-94*, 337-343, Seattle, WA (1994).
- Walser, J.P.; Solving Linear Pseudo-Boolean Constraint Problems with Local Search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI-97*, Providence, RI, (1997).
- Weld, D; Recent Advances in AI Planning. *AI Magazine*, (1999).
- Zhang, W; Swanson, H; and Moran, M.P; Modeling and Analyzing Soft Constraint Optimization Problems: Block-Exclusive Resource Allocation as a Case Study. *CP* (2001).