Appears in
Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-2001),
Seattle, WA, August 2001.

The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming

Dale Schuurmans and **Finnegan Southey**
Department of Computer Science
University of Waterloo
{dale,fdjsouth}@cs.uwaterloo.ca

Robert C. Holte*
Department of Computing Science
University of Alberta
holte@cs.ualberta.ca

Abstract

Boolean linear programs (BLPs) are ubiquitous in AI. Satisfiability testing, planning with resource constraints, and winner determination in combinatorial auctions are all examples of this type of problem. Although increasingly well-informed by work in OR, current AI research has tended to focus on specialized algorithms for each type of BLP task and has only loosely patterned new algorithms on effective methods from other tasks. In this paper we introduce a single general-purpose local search procedure that can be simultaneously applied to the entire range of BLP problems, without modification. Although one might suspect that a general-purpose algorithm might not perform as well as specialized algorithms, we find that this is not the case here. Our experiments show that our generic algorithm simultaneously achieves performance comparable with the state of the art in satisfiability search and winner determination in combinatorial auctions—two very different BLP problems. Our algorithm is simple, and combines an old idea from OR with recent ideas from AI.

1 Introduction

A Boolean linear program is a constrained optimization problem where one must choose a set of binary assignments to variables $x_1, ..., x_n$ to satisfy a given set of m linear inequalities $\mathbf{c}_1 \cdot \mathbf{x} \leq b_1, ..., \mathbf{c}_m \cdot \mathbf{x} \leq b_m$ while simultaneously optimizing a linear side objective $\mathbf{a} \cdot \mathbf{x}$. Specifically, we consider BLP problems of the canonical form

$$\min_{\mathbf{x} \in \{-1,+1\}^n} \mathbf{a} \cdot \mathbf{x} \quad \text{subject to} \quad C\mathbf{x} \leq \mathbf{b} \tag{1}$$

Clearly this is just a special case of integer linear programming (ILP) commonly referred to as 0-1 integer programming.[1] Many important problems in AI can be naturally expressed as BLP problems; for example: finding a satisfying truth assignment for CNF propositional formulae (SAT) [Kautz and Selman, 1996], winner determination in combinatorial auctions (CA) [Sandholm, 1999; Fujishima *et al.*, 1999], planning with resource constraints [Kautz and Walser, 1999; Vossen *et al.*, 1999], and scheduling and configuration problems [Walser, 1999; Lau *et al.*, 2000]. The two specific problems we will investigate in detail below are SAT and CA.

In general, BLP problems are hard in the worst case (NP-hard as optimization problems, NP-complete as decision problems). Nevertheless, they remain important problems and extensive research has been invested in improving the exponential running times of algorithms that guarantee optimal answers, developing heuristic methods that approximate optimal answers, and identifying tractable subcases and efficient algorithms for these subcases. There is a fundamental distinction between *complete* methods, which are guaranteed to terminate and produce an optimal solution to the problem, and *incomplete* methods, which make no such guarantees but nevertheless often produce good answers reasonably quickly. Most complete methods conduct a systematic search through the variable assignment space and use pruning rules and ordering heuristics to reduce the number of assignments visited while preserving correctness. Incomplete methods on the other hand typically employ local search strategies that investigate a small neighborhood of a current variable assignment (by flipping one or a small number of assignments) and take greedy steps by evaluating neighbors under a heuristic evaluation function. Although complete methods might appear to be more principled, incomplete search methods have proven to be surprisingly effective in many domains, and have led to significant progress in some fields of research (most notably on SAT problems [Kautz and Selman, 1996] but more recently on CA problems as well [Hoos and Boutilier, 2000]).

In this paper we investigate incomplete local search methods for general BLP problems. Although impressive, most of the recent breakthroughs in incomplete local search have been achieved by tailoring methods to a reduced class of BLP problems. A good illustration of this point is the CA system of Hoos and Boutilier [2000] (Casanova) which is based on the SAT system of [Hoos, 1999; McAllester *et al.*, 1997] (Novelty+), but is not the same algorithm (neither of these algorithms can be directly applied to the opposing problem). Another example is the WSAT(OIP) system of Walser [1999] which is an extension of WSAT [Selman *et al.*, 1994] and achieves impressive results on general ILP problems, but nev-

*Work performed while at the University of Ottawa.

[1] Although one could alternatively restrict the choice of values to $\{0, 1\}$ we often find it more convenient to use $\{-1, +1\}$.

ertheless requires soft constraints to be manually created to replace a given optimization objective.

Our main contribution is the following: Although a variety of research communities in AI have investigated BLP problems and developed effective methods for certain cases, the current level of specialization might not be yielding the benefits that one might presume. To support this observation we introduce a generic local search algorithm that when applied to specialized forms of BLP (specifically SAT and CA) achieves performance that competes with the state of the art on these problems. Our algorithm is based on combining a simple idea from OR (subgradient optimization for Lagrangian relaxation) with a recent idea from AI, specifically from machine learning theory (multiplicative updates and exponentiated gradients). The conclusion we draw is that research on general purpose methods might still prove fruitful, and that known ideas in OR might still yield additional benefits in AI problems beyond those already acknowledged.

2 Background

Research on constrained optimization in AI has become increasingly well informed by extensive foundational work in OR [Gomes, 2000]. OR has investigated the specific task of ILP in greater depth than AI, and has tended to place more emphasis on developing general purpose methods applicable across the entire range of ILP problems. Complete methods in OR typically employ techniques such as branch and bound (using linear programming or Lagrangian relaxation), cutting planes, and branch and cut to prune away large portions of the assignment space in a systematic search [Martin, 1999]. This research has yielded sophisticated and highly optimized ILP solvers, such as the commercial CPLEX system, which although general purpose, performs very well on the specialized problems investigated in AI. In fact, it is still often unclear whether specialized AI algorithms perform better [Andersson *et al.*, 2000].

Perhaps less well known to AI researchers is that beyond systematic search with branch and bound, the OR literature also contains a significant body of work on *incomplete* (heuristic) local search methods for approximately solving ILP problems; see for example [Magazine and Oguz, 1984; Larsson *et al.*, 1996]. The simplest and most widespread of these ideas is to use *subgradient optimization* (which we describe below).

Our algorithm arises from the observation that the most recent strategies developed for the SAT problem have begun to use an analog of subgradient optimization as their core search strategy; in particular the DLM system of [Wu and Wah, 2000] and the SDF system of [Schuurmans and Southey, 2000] (see also [Thornton and Sattar, 1999; Frank, 1997; Davenport *et al.*, 1994]). Interestingly, these are among the most effective methods for finding satisfying assignments for CNF formulae, and yet they appear to be recapitulating a forty year old idea in OR going back to [Everett, 1963]. Below we show that, indeed, a straightforward subgradient optimization approach (with just three basic improvements from AI) yields a state of the art SAT search strategy that competes with DLM (arguably the fastest SAT search procedure currently known).

Interestingly, the method we develop is not specialized to SAT problems in any way. In fact, the algorithm is a general purpose BLP search technique that can in principle be applied to any problem in this class. To demonstrate this point we apply the method *without modification* (beyond parameter setting) to CA problems and find that the method still performs well relative to the state of the art (although somewhat less impressively than on SAT problems). In this case we also find that the commercial CPLEX system performs very well and is perhaps the best of the methods that we investigated. Our results give us renewed interest in investigating general purpose algorithms for BLP that combine well understood methods from OR with recent ideas from AI.

3 The exponentiated subgradient algorithm

To explain our approach we first need to recap some basic concepts from constrained optimization theory. Consider the canonical BLP (1). For any constrained optimization problem of this form the *Lagrangian* is defined to be

$$L(\mathbf{x}, \mathbf{y}) \quad = \quad \mathbf{a} \cdot \mathbf{x} + \sum_{i=1}^{m} y_i (\mathbf{c}_i \cdot \mathbf{x} - b_i) \qquad (2)$$

where $\mathbf{c}_i$ is the ith row vector of the constraint matrix C and y_i is the real valued Lagrange multiplier associated with constraint c_i. One can think of the multipliers y_i simply as weights on the constraint violations $v_i \triangleq \mathbf{c}_i \cdot \mathbf{x} - b_i$. Thus the Lagrangian can be thought of as a penalized objective function where for a given vector of constraint violation weights one could imagine minimizing the penalized objective $L(\mathbf{x}, \mathbf{y})$. In this way, the Lagrangian turns a constrained optimization problem into an unconstrained optimization problem. In fact, the solutions of these unconstrained optimizations are used to define the *dual function*

$$D(\mathbf{y}) \quad = \quad \min_{\mathbf{x} \in \{-1,+1\}^n} L(\mathbf{x}, \mathbf{y}) \qquad (3)$$

The *dual problem* for (1) is then defined to be

$$\max_{\mathbf{y}} D(\mathbf{y}) \quad \text{subject to} \quad \mathbf{y} \geq \mathbf{0} \qquad (4)$$

Let D^* denote the maximum value of (4) and let P^* denote the minimum value of (1). Note that these are all just definitions. The reason that (4) can be considered to be a dual to (1) is given by the *weak duality theorem*, which asserts that $D^* \leq P^*$ [Bertsekas, 1995; Martin, 1999] (see Appendix A). Therefore, solving $\min_{\mathbf{x} \in \{-1,+1\}^n} L(\mathbf{x}, \mathbf{y})$ for any Lagrange multiplier vector $\mathbf{y} \geq \mathbf{0}$ gives a *lower bound* on the optimum value of the original problem (1). The best achievable lower bound is achieved by solving the dual problem of maximizing $D(\mathbf{y})$ subject to $\mathbf{y} \geq \mathbf{0}$, yielding the optimal value D^*. The difference $P^* - D^*$ is called the *duality gap*. A fundamental theorem asserts that there is no duality gap for linear (or convex) programs with real valued variables [Bertsekas, 1995], so in principle these problems can be solved by dual methods alone. However, this is no longer the case once the variables are constrained to be integer or $\{-1, +1\}$ valued.

Although the dual problem cannot be used to directly solve (1) because of the existence of a possible duality gap, obtaining lower bounds on the optimal achievable value can still

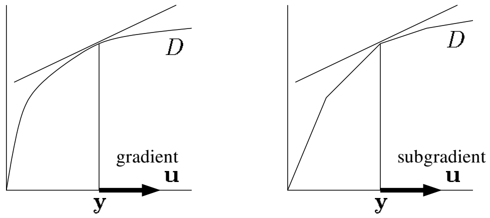

Figure 1: Subgradients generalize gradients to nondifferentiable functions. Note that the subgradient is not uniquely defined at a nondifferentiable point.

prove very useful in various search strategies, ranging from branch and bound to local search. Obviously there is a natural desire to maximize $D(\mathbf{y})$ by searching through Lagrange multiplier vectors for larger values. A natural way to proceed would be to start with a vector $\mathbf{y}^{(0)}$, solve the unconstrained minimization problem (3), and then update $\mathbf{y}^{(0)}$ to obtain a better weight vector $\mathbf{y}^{(1)}$, and so on. The issue is knowing how to update the weight vector $\mathbf{y}^{(0)}$. This question is complicated by the fact that $D(\mathbf{y})$ is typically nondifferentiable, which leads to the last technical development we will consider: *subgradient optimization*.

Since $D(\mathbf{y})$ is always known to be concave [Bertsekas, 1995], a *subgradient* of D (at a point $\mathbf{y}$) is given by any direction vector $\mathbf{u}$ such that $D(\mathbf{z}) \leq D(\mathbf{y}) + (\mathbf{z} - \mathbf{y})^\top \mathbf{u}$ for all $\mathbf{z} \in I\!R^m$. (Intuitively, a subgradient vector $\mathbf{u}$ gives the increasing direction of a plane that sits above D but touches D at $\mathbf{y}$, and hence can serve as a plausible search direction if one wishes to increase D; see Figure 1.) Therefore, to update $\mathbf{y}$, all we need to do is find a subgradient direction. Here we can exploit an extremely useful fact: after minimizing $L(\mathbf{x}, \mathbf{y})$ to obtain $\mathbf{x}_\mathbf{y} = \arg\min_{\mathbf{x} \in \{-1,+1\}^n} \mathbf{a} \cdot \mathbf{x} + \mathbf{y}^\top (C\mathbf{x} - \mathbf{b})$ *the resulting vector of residual constraint violation values* $\mathbf{v}_\mathbf{y} = C\mathbf{x}_\mathbf{y} - \mathbf{b}$ *is always a subgradient of* D *at* $\mathbf{y}$ [Bertsekas, 1995; Martin, 1999] (see Appendix B). So, in the end, despite the overbearing terminology, subgradient optimization is a fundamentally simple procedure:

Subgradient optimization: To improve the lower bound $D(\mathbf{y})$ on the optimal solution of the original problem, take a given vector of Lagrange multipliers $\mathbf{y}$, solve for a primal vector $\mathbf{x} \in \{-1, +1\}^n$ that minimizes the Lagrangian $L(\mathbf{x}, \mathbf{y})$, and update $\mathbf{y}$ by adding a proportional amount of the residual constraint violations $\mathbf{v}_\mathbf{y} = C\mathbf{x}_\mathbf{y} - \mathbf{b}$ to $\mathbf{y}$ (maintaining $\mathbf{y} \geq \mathbf{0}$); i.e. use the rule $\mathbf{y}' = \mathbf{y} + \alpha \mathbf{v}_\mathbf{y}$. Note that this has the intuitive effect of increasing the weights on the violated constraints while decreasing weights on satisfied constraints. Although this update is not guaranteed to increase the value of $D(\mathbf{y})$ at every iteration, it is guaranteed to move $\mathbf{y}$ closer to $\mathbf{y}^* = \arg\max_{\mathbf{y} \geq \mathbf{0}} D(\mathbf{y})$ for sufficiently small step-sizes α [Bertsekas, 1995].

These ideas go back at least to [Everett, 1963], and have been applied with great success by Held and Karp [1970] and many since. Typically, subgradient optimization has been used as a technique for generating good lower bounds for branch and bound methods (where it is known as *Lagrangian relaxation* [Fisher, 1981]). However, subgradient optimization can also be used as a heuristic search method for approximately solving (1) in a very straightforward way: at each iteration simply check if $\mathbf{x}_\mathbf{y}$ is feasible (i.e. satisfies $C\mathbf{x}_\mathbf{y} \leq \mathbf{b}$), and, if so, report $\mathbf{a} \cdot \mathbf{x}_\mathbf{y}$ as a feasible objective value (keeping it if it is the best value reported so far); see for example [Magazine and Oguz, 1984; Larsson *et al.*, 1996].

Interestingly, in the last few years this procedure has been inadvertently rediscovered in the AI literature. Specifically, in the field of incomplete local search methods for SAT, clause weighting schemes turn out to be using a form of subgradient optimization as their main control loop. These procedures work by fixing a profile of clause weights (Lagrange multipliers), greedily searching through variable assignment space for an assignment that minimizes a weighted score of clause violations (the Lagrangian), and then updating the clause weights by increasing them on unsatisfied clauses—all of which comprises a single subgradient optimization step. However, there are subtle differences between recent SAT procedures and the basic subgradient optimization approach outlined above. The DLM system of [Wu and Wah, 2000] explicitly uses a Lagrangian, but the multiplier updates follow a complex system of ad hoc calculations.[2] The SDF system of [Schuurmans and Southey, 2000] is simpler (albeit slower), but includes several details of uncertain significance. Nevertheless, the simplicity of this latter approach offers useful hypotheses for our current investigation.

Given the clear connection between recent SAT procedures and the traditional subgradient optimization technique in OR, we conduct a study of the deviations from the basic OR method so that we can identify the deviations which are truly beneficial. Our intent is to validate (or refute) the significance of some of the most recent ideas in the AI SAT literature.

Linear versus nonlinear penalties: One difference between recent SAT methods and subgradient optimization is that the SAT methods only penalize constraints with positive violations (by increasing the weight on these constraints), whereas standard subgradient optimization also rewards satisfied constraints (by reducing their weight proportionally to the degree of negative violation). To express this difference, consider an augmented Lagrangian which extends (2) by introducing a *penalty function* θ on constraint violations

$$L_\theta(\mathbf{x}, \mathbf{y}) \quad = \quad \mathbf{a} \cdot \mathbf{x} + \sum_{i=1}^m y_i \ \theta(\mathbf{c}_i \cdot \mathbf{x} - b_i)$$

The penalty functions we consider are the traditional linear penalty $\theta(v) = v$ and the "hinge" penalty

$$\theta(v) \quad = \quad \begin{cases} -\frac{1}{2} & \text{if } v \leq 0 \\ v - \frac{1}{2} & \text{if } v > 0 \end{cases}$$

(Note that v is an integer.) DLM and SDF can be interpreted as implicitly using a hinge penalty for dual updates on SAT problems. (However, SDF uses a different penalty for its primal search.) Intuitively, the hinge penalty has advantages because it does not favor increasing the satisfaction level of

[2] The Lagrangian used in [Wu and Wah, 2000] is also quite different from (2) because it includes a redundant copy of the constraint violations in the minimization objective. This prevents their Lagrangian from being easily generalized beyond SAT.

satisfied constraints above reducing the violations of unsatisfied constraints. Below we observe that the traditional linear penalty leads to poor performance on constraint satisfaction tasks and the AI methods have an advantage in this respect.

Unfortunately, the consequence of choosing a nonlinear penalty function is that finding a variable assignment $\mathbf{x}$ which minimizes $L_\theta(\mathbf{x}, \mathbf{y})$ is no longer tractable. To cope with this difficulty AI SAT solvers replace the global optimization process with a greedy local search (augmented with randomization). Therefore, they only follow a *local* subgradient direction in $\mathbf{y}$ at each update. However, despite the locally optimal nature of the dual updates, the hinge penalty appears to retain an advantage over the linear penalty.

Multiplicative versus additive updates: The SDF procedure updates $\mathbf{y}$ multiplicatively rather than additively, in an analogy to the work on multiplicative updates in machine learning theory [Kivinen and Warmuth, 1997]. A multiplicative update is naturally interpreted as following an exponentiated version of the subgradient; that is, instead of using the traditional additive update $\mathbf{y}' = \mathbf{y} + \alpha\boldsymbol{\theta}(\mathbf{v})$ one uses $\mathbf{y}' = \mathbf{y}\alpha^{\boldsymbol{\theta}(\mathbf{v})}$, $\alpha > 1$, given the vector of penalized violation values $\boldsymbol{\theta}(\mathbf{v})$. Below we compare both types of update and find that the multiplicative approach typically works better.

Weight smoothing: [Schuurmans and Southey, 2000] also introduce the idea of weight smoothing: after each update $\mathbf{y}' = \mathbf{y}\alpha^{\boldsymbol{\theta}(\mathbf{v})}$ the constraint weights are pulled back toward the population average $\overline{\mathbf{y}'} = \frac{1}{m}\sum_{i=1}^m y_i'$ using the rule $\mathbf{y}'' = \rho\mathbf{y}' + (1-\rho)\overline{\mathbf{y}'}$, $0 < \rho \le 1$. This has the effect of increasing the weights of frequently satisfied constraints without requiring them to be explicitly violated (which led to a noticeable improvement in SDF's performance).

Exponentiated subgradient optimization (ESG): The final procedure we propose follows a standard subgradient optimization search with three main augmentations: *(1)* we use the augmented Lagrangian L_θ with a hinge penalty rather than a linear penalty, *(2)* we use multiplicative rather than additive updates, and *(3)* we use weight smoothing to uniformize weights without requiring constraints to be explicitly violated. The parameters of the final procedure are α, ρ, and a noise parameter η (see Figure 2).[3] Below we compare four variants of the basic method: ESG_h, ESG_ℓ (multiplicative updates with hinge and linear penalties respectively), and ASG_h, ASG_ℓ (additive updates with each penalty).[4]

The final ESG procedure differs from the SDF system of [Schuurmans and Southey, 2000] in many respects. The most important difference is that ESG can be applied to arbitrary BLP tasks, whereas SDF is applicable only to SAT problems. However, even if a generalized form of SDF could be devised, there would still remain minor differences: First, ESG uses a constant multiplier α for dual updates, whereas SDF uses an adaptive multiplier that obtains a minimum difference δ in $L(\mathbf{x}, \mathbf{y})$ for some primal search direction. Second, SDF uses a different penalty in its primal and dual phases (exponential versus hinge) whereas ESG uses the same hinge penalty in both phases. Third, ESG smooths all of the weights, whereas SDF only smooths weights on satisfied constraints. Finally, ESG includes a small probability of stepping out of local minima during its primal search, whereas SDF employs a deterministic primal search. Overall, these simplifications allow for a more streamlined implementation for ESG that yields a significant reduction in CPU overhead per step, while simultaneously allowing application to more general tasks.

[3]Software available at http://ai.uwaterloo.ca/~dale/software/esg/.

[4]Note that when using a linear penalty we employ an optimal primal search, but when using a hinge penalty we must resort to a greedy local search to approximately minimize $L(\mathbf{x}, \mathbf{y})$.

ESG$_\theta(\alpha, \rho, \eta)$ **procedure**
Initialize $\mathbf{y} := \mathbf{1}$ and $\mathbf{x} :=$ random $\{-1, +1\}^n$
while solution not found **and** restart not reached
Primal search: Use greedy local search from $\mathbf{x}$ to find $\mathbf{x}'$ that locally minimizes $L_\theta(\mathbf{x}', \mathbf{y})$ (iff stuck, with probability η take a random move and continue); $\mathbf{v}' := C\mathbf{x}' - \mathbf{b}$.
Dual step: $\mathbf{y}' := \mathbf{y}\alpha^{\boldsymbol{\theta}(\mathbf{v}')}$; $\mathbf{y}'' := \rho\mathbf{y}' + (1-\rho)\overline{\mathbf{y}'}$.
Update: $\mathbf{x} := \mathbf{x}'$; $\mathbf{y} := \mathbf{y}''$

Figure 2: ESG procedure

4 SAT experiments

We consider applying the BLP techniques to SAT problems. An instance of SAT is specified by a set of clauses $\mathbf{c} = \{c_i\}_{i=1}^m$, where each clause c_i is a disjunction of k_i literals, and each literal denotes whether an assignment of "true" or "false" is required of a variable x_j. The goal is to find an assignment of truth values to variables such that every clause is satisfied on at least one literal. In our framework, an instance of SAT can be equivalently represented by the BLP

$$\min_{\mathbf{x}\in\{-1,+1\}^n} \mathbf{0}\cdot\mathbf{x} \quad \text{subject to} \quad C\mathbf{x} \le \mathbf{k} - \mathbf{2} \tag{5}$$

where $\mathbf{0}$ and $\mathbf{2}$ are vectors of zeros and twos respectively, and

$$c_{ij} = \begin{cases} 1 & \text{if } x_j \text{ appears in a negative literal in } c_i \\ -1 & \text{if } x_j \text{ appears in a positive literal in } c_i \\ 0 & \text{otherwise} \end{cases}$$

An assignment of $+1$ to variable x_j denotes "true", and an assignment of -1 denotes "false". The idea behind this representation is that a clause c_i is encoded by a row vector $\mathbf{c}_i$ in C which has k_i nonzero entries corresponding to the variables occurring in c_i, along with their signs. A constraint is violated only when the $\{-1, +1\}$ assignment to $\mathbf{x}$ agrees with the coefficients in $\mathbf{c}_i$ on every nonzero entry, yielding a row sum of $\mathbf{c}_i \cdot \mathbf{x} = k_i$. If a single sign is flipped then the row sum drops by 2 and the constraint becomes satisfied.

Note that the zero vector means that the minimization component of this problem is trivialized. Nevertheless it remains a hard constraint satisfaction problem. The trivialized objective causes no undue difficulty to the methods introduced above, and therefore the BLP formulation allows us to accommodate both constraint satisfaction and constrained optimization problems in a common framework (albeit in a more restricted way than [Hooker *et al.*, 1999]).

For this problem we compared the subgradient optimization techniques ESG/ASG against state of the art local search methods: DLM, SDF, Novelty+, Novelty, WSAT, GSAT and HSAT. However, for reasons of space we report results only

	Avg. Steps	Est. Opt. Steps	Fail %	Opt. sec
uf50	(100 problems)			
CPLEX (10 probs)	49	na	0	.186
ASG$_\ell(.12, 0, .02)$	500,000	na	100	na
ESG$_\ell(2.0, .05, .125)$	178,900	143,030	25	43.3
ASG$_h(.12, 0, .02)$	1,194	359	0.3	.027
ESG$_h(1.2, .99, .0005)$	215	187	0	.0009
uf100	(1000 problems)			
CPLEX (10 probs)	727	na	0	6.68
ASG$_h(.2, 0, .02)$	3,670	2,170	1.3	0.26
ESG$_h(1.15, .01, .002)$	952	839	0	.004
uf150	(100 problems)			
CPLEX (10 probs)	13,808	na	0	275.2
ASG$_h(.5, 0, .02)$	14,290	6,975	1.1	.071
ESG$_h(1.15, .01, .001)$	2,625	2,221	0	.011

Table 1: Comparison of general BLP methods on SAT

for DLM, SDF, and Novelty+, which represent the best of the last two generations of local search methods for SAT.[5] We ran these systems on a collection of (satisfiable) benchmark problems from the SATLIB repository.[6] Here the SAT solvers were run on their standard CNF input representations whereas the BLP solvers were run on BLP transformed versions of the SAT problems, as given above. (Although alternative encodings of SAT problems could affect the performance of BLP solvers [Beacham *et al.*, 2001] we find that the simple transformation described above yields reasonable results.)

To measure the performance of these systems we recorded wall clock time on a PIII 750MHz processor, average number of *primal* search steps ("flips") needed to reach a solution, and the proportion of problems not solved within 500K steps (5M on bw_large.c). To avoid the need to tune every method for a restart threshold we employed the strategy of [Schuurmans and Southey, 2000] and ran each method 100 times on every problem to record the distribution of solution times, and used this to estimate the minimum expected number of search steps required under an optimal restart scheme for the distribution [Parkes and Walser, 1996]. The remaining parameters of each method were manually tuned for every problem set. The parameters considered were: Novelty+(*walk, noise*), DLM(16 tunable parameters), SDF(δ, $1-\rho$), ESG$_\theta(\alpha, 1-\rho, \eta)$ and ASG$_\theta(\alpha, 1-\rho, \eta)$.[7]

[5] Implementations of these methods are available at ai.uwaterloo.ca/∼dale, www.satlib.org, and manip.crhc.uiuc.edu.

[6] SATLIB is at www.satlib.org. We used three classes of problems in our experiments. The *uf* problems are randomly generated 3CNF formulae that are generated at the phase transition ratio of 4.3 clauses to variables. (These formulae have roughly a 50% chance of being satisfiable, but *uf* contains only verified satisfiable instances.) The *flat* problems are SAT encoded instances of hard random graph coloring problems. The remaining problems are derived from AI planning problems and the "all interval series" problem.

[7] For the large random collections (Tables 1 and 2: flat100–uf250) the methods were tuned by running on 10 arbitrary problems. Performance was then measured on the complete collection. Novelty+ parameters were tuned from the published values of [Hoos and Stützle, 2000]. DLM was tuned by running each of the 5 default parameter sets available at manip.crhc.uiuc.edu and choosing the best.

	Avg. Steps	Est. Opt. Steps	Fail %	Opt. sec
ais8	(1 problem)			
Novelty+$(.4, .01)$	183,585	20,900	9	.078
SDF$(.0004, .005)$	4,490	4,419	0	.083
DLM(pars1)	4,678	4,460	0	.044
ESG$_h(1.9, .001, .0006)$	4,956	4,039	0	.043
ais10	(1 problem)			
Novelty+$(.3, .01)$	434,469	340,700	79	1.64
SDF$(.00013, .005)$	15,000	13,941	0	.40
DLM(pars4)	18,420	14,306	0	.11
ESG$_h(1.9, .001, .0004)$	15,732	15,182	0	.23
ais12	(1 problem)			
Novelty+$(.5, .01)$	496,536	496,536	99	3.0
SDF$(.0001, .005)$	132,021	97,600	1	3.6
DLM(pars1)	165,904	165,904	3	2.5
ESG$_h(1.8, .001, .0005)$	115,836	85,252	10	1.7
bw_large.a	(1 problem)			
Novelty+$(.4, .01)$	9,945	8,366	0	.025
SDF$(.0001, .005)$	3,012	3,012	0	.057
DLM(pars4)	3,712	3,701	0	.028
ESG$_h(3, .005, .0015)$	2,594	2,556	0	.022
bw_large.b	(1 problem)			
Novelty+$(.4, .01)$	210,206	210,206	12	.82
SDF$(.00005, .005)$	33,442	33,255	0	1.30
DLM(pars4)	44,361	39,216	0	.34
ESG$_h(1.4, .01, .0005)$	33,750	26,159	0	.40
bw_large.c	(1 problem)			
Novelty+$(.2, .01)$	3,472,770	1,350,620	52	7
SDF$(.00002, .005)$	3,478,770	3,478,770	48	313
DLM(pars4)	3,129,450	2,282,900	32	41
ESG$_h(1.4, .01, .0005)$	1,386,050	875,104	5	39
logistics.c	(1 problem)			
Novelty+$(.4, .01)$	127,049	126,795	1	.38
SDF$(.000009, .005)$	15,939	15,939	0	1.40
DLM(pars4)	12,101	11,805	0	.13
ESG$_h(2.2, .01, .0025)$	8,962	8,920	0	.15
flat100	(100 problems)			
Novelty+$(.6, .01)$	17,266	12,914	.03	.019
SDF$(.0002, .005)$	7,207	6,478	0	.074
DLM(pars4)	9,571	8,314	0	.022
ESG$_h(1.1, .01, .0015)$	7,709	6,779	0	.022
flat200	(100 problems)			
Novelty+$(.6, .01)$	238,663	218,274	27	.34
SDF$(.0001, .005)$	142,277	115,440	7	1.56
DLM(pars4)	280,401	242,439	31	.77
ESG$_h(1.01, .01, .0025)$	175,721	134,367	14	.47
uf150	(100 problems)			
Novelty+$(.6, .01)$	6,042	3,970	0	.008
SDF$(.00065, .005)$	3,151	2,262	0	.023
DLM(pars4)	3,263	2,455	0	.008
ESG$_h(1.15, .01, .001)$	2,625	2,221	0	.011
uf200	(100 problems)			
Novelty+$(.6, .01)$	25,917	22,214	1.8	.048
SDF$(.0003, .005)$	14,484	11,304	.4	.134
DLM(pars4)	13,316	9,020	.1	.030
ESG$_h(1.23, .01, .003)$	10,583	7,936	.2	.039
uf250	(100 problems)			
Novelty+$(.6, .01)$	27,730	24,795	1.8	.055
SDF$(.0002, .005)$	18,404	13,626	.1	.177
DLM(pars4)	22,686	12,387	.2	.042
ESG$_h(1.15, .01, .003)$	13,486	10,648	.1	.054

Table 2: Comparison of ESG$_h$ to specialized SAT methods

Table 1 compares the different BLP procedures on random problems from SATLIB. The most salient feature of these results is that the linear penalty performs poorly (as anticipated). Here, the traditional ASG_ℓ method was unable to solve any problems in the allotted steps. Table 1 also shows that multiplicative updates (ESG) were generally superior to additive updates (ASG) regardless of the penalty function used (at least on random SAT problems). Although details are omitted, we have also found that some smoothing $(1 - \rho \leq 0.01)$ usually improves the performance of ESG but is generally less beneficial for ASG. CPLEX generally performs poorly in these experiments.

Table 2 shows the results of a larger scale comparison between ESG_h and state of the art local search methods for SAT. These results show that ESG_h tends to finds solutions in fewer primal search steps (flips) than other procedures. However, ESG_h's time per flip is about 1.5 to 2 times higher than DLM and about 2 to 4 times higher than Novelty+, which means that its overall run time is only comparable to these methods. Nevertheless, ESG_h, DLM and SDF obtain a large reduction in search steps over Novelty+ on the structured ais and planning problems (with the notable exception of bw_large.c). As a result, DLM and ESG_h both tend to demonstrate better run times than Novelty+ on these problems. All three methods (Novelty+, DLM and ESG_h) demonstrate similar run times on the random uf and flat problems.

Note that none of these SAT procedures (other than ESG/ASG and CPLEX) can be applied to general BLP problems without modification.

5 CA experiments

The second problem we considered is optimal winner determination in combinatorial auctions (CA). This problem introduces a nontrivial optimization objective that is not present in SAT. However, the subgradient optimization approach remains applicable, and we can apply the same ESG/ASG methods to this task without modifying them in any way. (Nevertheless, we did conduct some implementation specializations to gain improvements in CPU times on SAT problems.) The CA problem has been much less studied in the AI literature, but interest in the problem is growing rapidly.

An instance of the optimal winner determination problem in combinatorial auctions (CA) is given by a set of items $\mathbf{c} = \{c_i\}_{i=1}^m$ with available quantities $\mathbf{q} = \{q_i\}_{i=1}^m$, and a set of bids $\mathbf{z} = \{z_j\}_{j=1}^n$ which offer amounts $\mathbf{v} = \{v_j\}_{j=1}^n$ for a specified subset of items. We can represent the bid requests in a constraint matrix C where

$$c_{ij} \quad = \quad \begin{cases} 1 & \text{if bid } z_j \text{ requests item } c_i \\ 0 & \text{otherwise} \end{cases}$$

The problem then is to find a set of bids that maximizes the total revenue subject to the constraint that none of the item quantities are exceeded. If $z_j \in \{0, 1\}$ this problem can be expressed as the BLP

$$\max_{\mathbf{z} \in \{0,1\}} \mathbf{v} \cdot \mathbf{z} \quad \text{subject to} \quad C\mathbf{z} \leq \mathbf{q} \tag{6}$$

However it is not in our canonical $\{-1, +1\}$ form. To transform it to the canonical form we use the substitutions $\mathbf{x} = 2\mathbf{z} - 1$, $\mathbf{a} = -\mathbf{v}/2$ and $\mathbf{b} = 2\mathbf{q} - C\mathbf{1}$. The minimum solution to the transformed version of this problem can then be converted back to a maximum solution of the original CA.

For this task we compared the ESG/ASG algorithms to CPLEX and Casanova [Hoos and Boutilier, 2000], a local search method loosely based on Novelty+.[8] The problems we tested on were generated by the CATS suite of CA problem generators [Leyton-Brown *et al.*, 2000], which are intended to model realistic problems. However, our results indicate that these tend to be systematically easy problems for all the methods we tested, so we also tested on earlier artificial problem generators from the literature [Sandholm, 1999; Fujishima *et al.*, 1999]. Some of these earlier generators were shown to be vulnerable to trivial algorithms [Andersson *et al.*, 2000] but some still appear to generate hard problems. However, to push the envelope of difficulty further, we also encoded several hard SAT problems as combinatorial auctions by using an efficient polynomial (quadratic) reduction from SAT to CA. Unfortunately, space limitations preclude a detailed description of this reduction, but our results show that these converted SAT problems are by far the most difficult available at a given problem size.

To measure the performance of the various methods, we first solved all of the problems using CPLEX and then ran the local search methods until they found an optimal solution or timed out. Although we acknowledge that this method of reporting ignores the anytime performance of the various methods, it seems sufficient for our purposes. To give some indication of anytime behavior, we recorded the fraction of optimality achieved by the local search methods in cases where they failed to solve the problem within the allotted time.

Table 3 shows a comparison of the different subgradient optimization procedures on one of the CATS problem classes. These results show that the linear penalty is once again weaker than the hinge (but to a lesser extent than on SAT problems). Interestingly, additive updates appear to work as well as multiplicative updates on these problems (although the performance of ASG begins to weaken on harder constraint systems such as those encountered in Table 5).

Table 4 shows the results of a larger comparison between ESG/ASG, Casanova and CPLEX on the CATS problems and an artificial problem. These results show that there is very little reason not to use CPLEX to solve these problems in practice.[9] CPLEX, of course, also proves that its answers are optimal, unlike local search. Nevertheless, the results show that ESG_h and ASG_h are roughly competitive with Casanova, which is a specialized local search technique for this task.

[8] Casanova is specialized to single unit CA problems ($\mathbf{q} = \mathbf{1}$), so we restrict attention to this class. However, ESG/ASG and CPLEX can be applied to multi-unit CA problems without modification, and hence are more general. CPLEX was run with the parameter settings reported in [Andersson *et al.*, 2000]. Casanova uses the same parameters as Novelty+, *walk* and *noise*.

[9] The CPLEX timings show time to first discover the optimum, not prove that it is indeed optimal. Also note that even though "steps" are recorded for each of the methods, they are incomparable numbers. Each method uses very different types of steps (unlike the previous SAT comparison in terms of "flips") and are reported only to show method-specific difficulty.

Note that the large score proportions obtained by all methods when they fail to find the optimal solution follows from the fact that even simple hill-climbing strategies tend to find near optimal solutions in a single pass [Holte, 2001]. It appears that most of the difficulty in solving these problems is in gaining the last percentage point of optimality.

Finally, Table 5 shows that Casanova demonstrates inferior performance on the more difficult SAT→CA encoded problems. Nevertheless all methods appear to be challenged by these problems. Interestingly, CPLEX slows down by a factor of 400 when transforming from the direct SAT encoding of Section 4 to the SAT→CA encoding used here. By comparison the ESG method slows down by a factor of 20K.

6 Conclusion

Although we do not claim that the methods we have introduced exhibit the absolute best performance on SAT or CA problems, they nevertheless compete very well with the state of the art techniques on both types of problem. Therefore, we feel that the simple Lagrangian approach introduced here might offer a useful starting point for future investigation beyond the myriad WalkSAT variants currently being pursued in AI. Among several directions for future work are to investigate other general search ideas from the OR literature, such as tabu search [Glover, 1989], and compare to other local search methods for ILP problems [Resende and Feo, 1996; Voudouris and Tsang, 1996] (which do not appear to be competitive on SAT problems, but still offer interesting perspectives on ILP problems in general).

A Weak duality

It is insightful to see why $D(\mathbf{y}) \leq P^*$ for all $\mathbf{y} \geq \mathbf{0}$. In fact, it is almost immediately obvious: Note that for any $\mathbf{y} \geq \mathbf{0}$ we have

$$\begin{aligned} D(\mathbf{y}) &= \min_{\mathbf{x} \in \{-1,+1\}^n} L(\mathbf{x}, \mathbf{y}) \\ &\leq \mathbf{a} \cdot \mathbf{x} + \mathbf{y}^\top (C\mathbf{x} - \mathbf{b}) \quad \text{for all } \mathbf{x} \in \{-1,+1\}^n \\ &\leq \mathbf{a} \cdot \mathbf{x} \quad \text{for all } \mathbf{x} \text{ such that } C\mathbf{x} \leq \mathbf{b} \quad (\text{since } \mathbf{y} \geq \mathbf{0}) \end{aligned}$$

and hence $D(\mathbf{y}) \leq P^*$. From this one can see that the constraint $\mathbf{y} \geq \mathbf{0}$ is imposed precisely to ensure a lower bound on P^*.

B Subgradient direction

It is also easy to see that the vector of constraint violation values $\mathbf{v}_\mathbf{y} = C\mathbf{x}_\mathbf{y} - \mathbf{b}$ (where $\mathbf{x}_\mathbf{y} = \arg\min_{\mathbf{x} \in \{-1,+1\}^n} L(\mathbf{x}, \mathbf{y})$) must yield a subgradient of D at $\mathbf{y}$:

$$\begin{aligned} D(\mathbf{z}) &= \min_{\mathbf{x} \in \{-1,+1\}^n} \mathbf{a} \cdot \mathbf{x} + \mathbf{z}^\top (C\mathbf{x} - \mathbf{b}) \\ &\leq \mathbf{a} \cdot \mathbf{x}_\mathbf{y} + \mathbf{z}^\top (C\mathbf{x}_\mathbf{y} - \mathbf{b}) \ \ (\text{since } L(\mathbf{x}_\mathbf{z}, \mathbf{z}) \leq L(\mathbf{x}_\mathbf{y}, \mathbf{z})) \\ &= \mathbf{a} \cdot \mathbf{x}_\mathbf{y} + \mathbf{y}^\top (C\mathbf{x}_\mathbf{y} - \mathbf{b}) + (\mathbf{z} - \mathbf{y})^\top (C\mathbf{x}_\mathbf{y} - \mathbf{b}) \\ &= D(\mathbf{y}) + (\mathbf{z} - \mathbf{y})^\top \mathbf{v}_\mathbf{y} \end{aligned}$$

Acknowledgments

Research supported by NSERC and CITO. Thanks to Holger Hoos and Craig Boutilier for helpful comments and providing access to Casanova. Thanks also to Peter van Beek and the anonymous referees for many helpful suggestions. The assistance of István Hernádvölgyi is also warmly appreciated.

	Avg. sec	Avg. Steps	Fail %	% Opt.
CATS-regions	(100 problems)			
$\text{ESG}_h(1.9, .1, .01)$	7.2	1416	4.1	99.93
$\text{ASG}_h(.045, 0, .01)$	12.7	2457	7.9	99.86
$\text{ESG}_\ell(1.3, .9, .01)$	64.7	7948	77	88.11
$\text{ASG}_\ell(.01, 0, .01)$	48.1	9305	90	93.96

Table 3: Comparison of subgradient optimization methods

	Avg. sec	Avg. Steps	Fail %	% Opt.
CATS-regions	(100 problems)			
CPLEX	6.7	64117	0	100
Casanova$(.5, .17)$	4.2	1404	3.4	99.95
$\text{ESG}_h(1.9, .1, .01)$	7.2	1416	4.1	99.93
$\text{ASG}_h(.045, 0, .01)$	12.7	2457	7.9	99.86
CATS-arbitrary	(100 problems)			
CPLEX	22	9510	0	100
Casanova$(.04, .12)$	9	2902	0.47	99.98
$\text{ESG}_h(2.1, .05, .5)$	33	7506	4.21	99.95
$\text{ASG}_h(.04, 0, .01)$	30	6492	4.87	99.87
CATS-matching	(100 problems)			
CPLEX	1.30	499	0	100
Casanova$(.3, .17)$	.17	109	0	100
$\text{ESG}_h(1.7, .05, 0)$	.16	215	0	100
$\text{ASG}_h(.9, 0, .8)$	.73	1248	0	100
CATS-paths	(100 problems)			
CPLEX	25	1	0	100
Casanova$(.5, .15)$	26	49	0	100
$\text{ESG}_h(1.3, .05, .4)$	28	2679	2.5	99.99
$\text{ASG}_h(.01, 0, .15)$	75	5501	6.8	99.96
CATS-scheduling	(100 problems)			
CPLEX	15	1426	0	100
Casanova$(.5, .04)$	44	7017	19.9	99.87
$\text{ESG}_h(1.35, .05, .015)$	65	11737	41	99.68
$\text{ASG}_h(.015, 0, .125)$	58	12925	44.2	99.51
Decay-200-200-.75	(100 problems)			
CPLEX	1.1	2014	0	100
Casanova$(.5, .17)$	0.5	2899	2	99.97
$\text{ESG}_h(14.5, .045, .45)$	1.8	24466	96.3	96.91
$\text{ASG}_h(.04, 0, .5)$	1.7	24939	99.6	91.49

Table 4: Results on CATS and synthetic problems

	Avg. sec	Avg. Steps	Fail %	% Opt.
SAT(uf50)→CA	(10 problems)			
CPLEX	42	754	0	100
Casanova$(.5, .11)$	468	9.6×10^6	90	99.36
$\text{ESG}_h(30, .05, .1)$	31	1.7×10^6	10	99.95
$\text{ASG}_h(5, 0, .5)$	173	8.6×10^6	80	99.40
SAT(uf75)→CA	(10 problems)			
CPLEX	666	5614	0	100
Casanova$(.5, .11)$	800	1.0×10^7	100	98.46
$\text{ESG}_h(41, .05, .1)$	165	6.2×10^6	60	99.81
$\text{ASG}_h(10, 0, .5)$	291	1.0×10^7	100	99.17

Table 5: Results on hard SAT→CA encoded problems

References

[Andersson *et al.*, 2000] A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings ICMAS-00*, 2000.

[Beacham *et al.*, 2001] A. Beacham, X. Chen, J. Sillito, and P. van Beek. Constraint programming lessons learned from crossword puzzles. In *Proc. Canadian AI Conf.*, 2001.

[Bertsekas, 1995] D. Bertsekas. *Nonlinear Optimization*. Athena Scientific, 1995.

[Davenport *et al.*, 1994] A. Davenport, E. Tsang, C. Wang, and K. Zhu. GENET: A connectionist architecture for solving constraint satisfaction problems. In *Proceedings AAAI-94*, pages 325–330, 1994.

[Everett, 1963] H. Everett. Generalized Lagrange multiplier method for solving problems of the optimal allocation of resources. *Operations Res.*, 11:399–417, 1963.

[Fisher, 1981] M. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Sci.*, 27:1–18, 1981.

[Frank, 1997] J. Frank. Learning short-tem weights for GSAT. In *Proceedings IJCAI-97*, pages 384–391, 1997.

[Fujishima *et al.*, 1999] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: optimal and approximate approaches. In *Proceedings IJCAI-99*, pages 548–553, 1999.

[Glover, 1989] F. Glover. Tabu search Part 1. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Gomes, 2000] C. Gomes. Structure, duality, and randomization: Common themes in AI and OR. In *Proceedings AAAI-00*, pages 1152–1158, 2000.

[Held and Karp, 1970] M. Held and R. Karp. The travelling salesman problem and minimum spanning trees. *Operations Res.*, 18:1138–1162, 1970.

[Holte, 2001] R. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings Canadian AI Conference*, 2001.

[Hooker *et al.*, 1999] J. Hooker, G. Ottosson, E. Thorsteinsson, and H.-K. Kim. On integrating constraint propagation and linear programming for combinatorial optimization. *Proceedings AAAI-99*, pages 136–141, 1999.

[Hoos and Boutilier, 2000] H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings AAAI-00*, pages 22–29, 2000.

[Hoos and Stützle, 2000] H. Hoos and T. Stützle. Local search algorithms for SAT: An empirical evaluation. *J. Automat. Reas.*, 24:421–481, 2000.

[Hoos, 1999] H. Hoos. On the run-time behavior of stochastic local search algorithms for SAT. In *Proceedings AAAI-99*, pages 661–666, 1999.

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

[Kautz and Walser, 1999] H. Kautz and J. Walser. State-space planning by integer optimization. *Proceedings AAAI-99*, pages 526–533, 1999.

[Kivinen and Warmuth, 1997] J. Kivinen and M. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Infor. Comput.*, 132:1–63, 1997.

[Larsson *et al.*, 1996] T. Larsson, M. Patriksson, and A.-B. Stromberg. Conditional subgradient optimization—theory and applications. *Euro. J. Oper. Res.*, 88:382–403, 1996.

[Lau *et al.*, 2000] H. Lau, A. Lim, and Q. Liu. Solving a supply chain optimization problem collaboratively. *Proceedings AAAI-00*, pages 780–785, 2000.

[Leyton-Brown *et al.*, 2000] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proc. EC-00*, 2000.

[Magazine and Oguz, 1984] M. Magazine and O. Oguz. A heuristic algorithm for the multidimensional knapsack problem. *Euro. J. Oper. Res.*, 16:319–326, 1984.

[Martin, 1999] R. Martin. *Large Scale Linear and Integer Optimization*. Kluwer, 1999.

[McAllester *et al.*, 1997] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings AAAI-97*, pages 321–326, 1997.

[Parkes and Walser, 1996] A. Parkes and J. Walser. Tuning local search for satisfiability testing. In *Proceedings AAAI-96*, pages 356–362, 1996.

[Resende and Feo, 1996] M. Resende and T. Feo. A GRASP for satisfiability. In *Cliques, Coloring, and Satisfiability*, DIMACS series v.26, pages 499–520. AMS, 1996.

[Sandholm, 1999] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings IJCAI-99*, pages 542–547, 1999.

[Schuurmans and Southey, 2000] D. Schuurmans and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proc. AAAI-00*, pages 297–302, 2000.

[Selman *et al.*, 1994] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings AAAI-94*, pages 337–343, 1994.

[Thornton and Sattar, 1999] J. Thornton and A. Sattar. On the behavior and application of constraint weighting. In *Proceedings CP-99*, pages 446–460, 1999.

[Vossen *et al.*, 1999] T. Vossen, M. Ball, A. Lotem, and D. Nau. On the use of integer programming models in AI planning. *Proceedings IJCAI-99*, pages 304–309, 1999.

[Voudouris and Tsang, 1996] C. Voudouris and E. Tsang. Partial constraint satisfaction problems and guided local search. In *Proceedings PACT-96*, pages 337–356, 1996.

[Walser, 1999] J. Walser. *Integer Optimization by Local Search*. Springer-Verlag, 1999.

[Wu and Wah, 2000] Z. Wu and W. Wah. An efficient global-search strategy in discrete Lagrangian methods for solving hard satisfiability problems. In *Proceedings AAAI-00*, pages 310–315, 2000.