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
School of Information Technology and Engineering
University of Ottawa
holte@site.uottawa.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

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

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]. Two specific problems we will investigate in detail below are SAT and CA. To demonstrate that these are indeed instances of Boolean programming problems, consider the explicit characterization of their representation as BLPs.

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

SAT: 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

$$\min_{\mathbf{x} \in \{-1,+1\}^n} \mathbf{0} \cdot \mathbf{x} \quad \text{subject to} \quad C\mathbf{x} \leq \mathbf{k} - \mathbf{2} \qquad (2)$$

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 notes "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 we discuss below, 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]). To illustrate this further, consider a nontrivial optimization problem.

CA: 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. Let $G_j \subset \mathbf{c}$ denote the set of items requested by bid z_j. We can represent the requests in a constraint matrix C where

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

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 a BLP

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

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.

In a similar fashion one can also represent planning, configuration and scheduling problems as BLPs, although this often requires more complex encodings than those illustrated above (for example, the representation of planning as an ILP problem presented in [Kautz and Walser, 1999]).

In general, BLP problems are hard in the worst case (NP-hard as optimization problems, NP-complete as decision problems). They nevertheless remain important tasks and extensive research has been invested in improving 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. A fundamental distinction exists 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, using pruning rules and ordering heuristics to reduce the number of assignments visited while maintaining 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, greedy local 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 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] (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 and achieves impressive results on general ILP problems, but nevertheless requires soft constraints to be 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.

OR has thoroughly investigated the ILP problem in much greater depth than AI. This literature 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].

What is 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]. 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]. Interestingly, these are among the most effective methods for finding satisfying assignments for CNF formulae, and yet they appear to be recapitulating a thirty year old idea in OR going back to [Everett, 1963].

One of our contributions is to show that, indeed, a straightforward subgradient optimization approach (with just two 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.

2 The exponentiated subgradient algorithm

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

$$L(\mathbf{x},\mathbf{y}) \;=\; \mathbf{a}\cdot\mathbf{x} + \sum_{i=1}^{m} y_i(\mathbf{c}_i\cdot\mathbf{x} - b_i) \qquad (3)$$

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 $v_i \overset{\triangle}{=} \mathbf{c}_i \cdot \mathbf{x} - b_i$. 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*

$$D(\mathbf{y}) \;=\; \min_{\mathbf{x}\in\{-1,+1\}^n} L(\mathbf{x},\mathbf{y}) \qquad (4)$$

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

$$\max_{\mathbf{y}} D(\mathbf{y}) \quad \text{subject to} \quad \mathbf{y} \geq \mathbf{0} \qquad (5)$$

Let D^* denote the maximum value of (5) and let P^* denote the minimum value of (1). Note that these are all just definitions. The reason that (5) 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]. 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 the lower bounds on the optimal achievable value can still prove very useful in various search strategies, ranging from branch and bound to local search. Clearly, 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 (4), 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 non-differentiable. Coping with non-differentiability leads to the last technical development we will consider: *subgradient optimization*.

Since $D(\mathbf{y})$ is always known to be concave [Bertsekas, 1995], the 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 tangent plane that sits above D at $\mathbf{y}$, and hence can serve as a plausible search direction if one wishes to increase D.) Therefore, to update $\mathbf{y}$, all we need to do is find a subgradient direction. Here is where 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]. 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* [Martin, 1999]). 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].

Interestingly, in the last few years this procedure has been inadvertently been rediscovered in the AI literature. Specifically, in the field of incomplete local search methods for SAT, clause weighting methods 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. However, the basic simplicity of this latter approach offers many interesting hypotheses for our investigation.

Given the clear connection between recent SAT procedures and the traditional subgradient optimization technique in OR, we conduct a controlled 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 the degree of negative violation). To express this difference, consider an augmented Lagrangian which extends (3) 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 simply 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.[3] Intuitively, the hinge penalty has advantages because it does not favor reducing the violation level of several satisfied constraints above reducing the violations of a few unsatisfied constraints. We verify below that the traditional linear penalty function leads to poor performance on constraint satisfaction tasks and the AI methods have a distinct advantage in this respect.

Unfortunately, the consequence of choosing a nonlinear penalty function is that finding a variable assignment 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 randomness). 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 retains a significant advantage over the linear penalty, which we verify below.

[2]The Lagrangian used in [Wu and Wah, 2000] is also quite different from (3) because it includes a redundant copy of the constraint violations in the minimization objective. However, this prevents their Lagrangian from being easily generalizable beyond SAT.

[3]SDF apparently uses a different penalty for its primal search.

Multiplicative versus additive updates: The SDF procedure updates $\mathbf{y}$ multiplicatively rather than additively, in an apparent 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 $\mathbf{y}' = \mathbf{y}\alpha^{\boldsymbol{\theta}(\mathbf{v})}$ given the vector of penalized violation values $\boldsymbol{\theta}(\mathbf{v})$. Below we compare additive and multiplicative updates and verify that multiplicative updates are often advantageous.

Exponentiated subgradient optimization (ESG): The final procedure we propose follows a standard standard subgradient optimization search with two main augmentations: *(1)* we use the augmented Lagrangian L_θ with a hinge penalty rather than a linear penalty, and *(2)* we use multiplicative rather than additive updates. An additional idea we exploit is the weight smoothing technique of [Schuurmans and Southey, 2000] where after each update $\mathbf{y}' = \mathbf{y}\alpha^{\boldsymbol{\theta}(\mathbf{v})}$ the weights are pulled back toward the population average $\overline{\mathbf{y}'} = \frac{1}{m}\sum_{i=1}^{m} y_i'$ using the rule $\mathbf{y}'' = \rho\mathbf{y}' + (1-\rho)\overline{\mathbf{y}'}$. (We have observed that smoothing indeed yields benefits for multiplicative updates.) To summarize, the final parameters of the final procedure are α, ρ, and a noise parameter η which determines how often a random move is made in the primal search phases. In our experiments below we compare four variants of the basic method: ESG$_h$, ESG$_\ell$ (multiplicative updates with hinge and linear penalties respectively), and ASG$_h$, ASG$_\ell$ (additive updates with each penalty).

3 SAT experiments

For this problem we compared the various subgradient optimization techniques introduced above against state of the art local search methods. Specifically we compared DLM, SDF, Novelty+, Novelty, WSAT, GSAT and HSAT. However, for reasons of space we report results only for DLM and Novelty+, which represent the best of the last two generations of local search methods for this problem. We ran the systems on a collection of (satisfiable) benchmark SAT problems in the SATLIB repository. To measure performance we recorded wall clock time (running on a PIII 750MHz cluster), average number of primal search steps (variable "flips") needed to reach an optimal solution, and the proportion of problems not solved within 500K steps (1M on bw_large.c).

To avoid the need to tune the methods for restarts we employed the strategy of [Schuurmans and Southey, 2000] and ran each method 100 times on each 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]. Table 1 summarizes our results.

The outcome basically supports the hypothesis that multiplicative updates (ESG) are more effective than additive updates (ASG), at least on SAT problems. Table 1 also shows that the traditional linear penalty performs very poorly (as anticipated). Moreover, the results show that ESG$_h$ systematically produces optimal answers in fewer primal search steps (flips) than other procedures, and achieves run times that are competitive with DLM and frequently better than Novelty+.

	Avg. Steps	Est. Optim. Steps	Fail %	Total msec /Step
flat100	(100 problems)			
Novelty+$(.5, .01)$	23,995	13,893	.2	2.9
DLM(pars4)	9,571	8,314	0	4.8
ESG$_h(1.1, .01, .0015)$	7,393	6,665	0	6.6
flat200	(100 problems)			
Novelty+$(.5, .01)$	221,257	192,575	22	3.2
DLM(pars4)	280,401	242,439	31	5.8
ESG$_h(1.01, .01, .0025)$	177,899	145,221	14	7.7
ais8	(1 problem)			
Novelty+$(.5, .01)$	169,626	137,436	8	9.9
DLM(pars4)	5376	5326	0	11
ESG$_h(1.9, .001, .0006)$	4956	4039	0	21
ais10	(1 problem)			
Novelty+$(.5, .01)$	451,222	273,300	84	14
DLM(pars4)	18,420	14,306	0	16
ESG$_h(1.9, .001, .0004)$	15,732	15,182	0	33
ais12	(1 problem)			
Novelty+$(.5, .01)$	497,518	497,518	99	19
DLM(pars4)	215,360	179,383	7	17
ESG$_h(1.8, .001, .0005)$	115,836	85,252	1	46
uf50	(10 problems)			
Novelty+$(.5, .01)$	217	217	0	4.3
DLM(pars4)	198	152	0	6.2
ESG$_h(1.2, .99, .0005)$	225	171	0	8.3
*ASG$_h(.12, .02)$	783	271	0.1	10.9
*ESG$_\ell(2.0, .95, .125)$	178900	143030	25	30.3
*ASG$_\ell(.12, .02)$	500000	na	100	20.1
uf150	(100 problems)			
Novelty+$(.5, .01)$	8331	4456	.03	4.5
DLM(pars4)	3263	2455	0	6.0
ESG$_h(1.15, .01, .001)$	2649	2300	0	10
*ASG$_h(.5, .02)$	17100	5978	2.6	74
uf200	(100 problems)			
Novelty+$(.5, .01)$	28,529	17,953	2.3	4.9
DLM(pars4)	12,215	9,020	.1	6.2
ESG$_h(1.23, .01, .003)$	10,360	8,947	.18	11
uf250	(100 problems)			
Novelty+$(.5, .01)$	31,560	21,434	2.2	4.9
DLM(pars4)	22,635	12,387	.3	6.6
ESG$_h(1.15, .01, .003)$	13,529	10,596	.14	12
bw_large.a	(1 problem)			
Novelty+$(.5, .01)$	10,788	10,323	0	8
DLM(pars4)	3712	3701	0	15
ESG$_h(3, .005, .0015)$	2594	2556	0	20
bw_large.b	(1 problem)			
Novelty+$(.5, .01)$	373,001	217,394	53	12
DLM(pars4)	44,361	39,216	0	18
ESG$_h(1.4, .01, .0005)$	33,750	26,159	0	33
bw_large.c	(1 problem)			
Novelty+$(.5, .01)$	1,000,000	na	100	21
DLM(pars4)	895,213	895,213	77	33
ESG$_h(1.4, .0005)$	695,565	690,565	63	64
logistics.c	(1 problem)			
Novelty+$(.5, .01)$	163,622	163,622	1	9
DLM(pars4)	12,101	11,805	0	25
ESG$_h(2.2, .01, .0025)$	8,962	8,920	0	35

Table 1: SAT results on benchmark SATLIB problems. Parameters: Novelty+(*walk*, *noise*), DLM(a 21 parameter set from http://manip.crhc.uiuc.edu), ESG$_h(\alpha, 1-\rho, noise)$, ASG$_h(\alpha, noise)$.

4 CA experiments

The second problem we investigated is optimal winner determination in combinatorial auctions (CA). This problem introduces a nontrivial optimization objective. However, the subgradient optimization approach remains applicable, and we can apply the same ESG/ASG methods to this problem without modifying them in any way. (However, 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.

	Avg. Time	Avg. Steps	Fail %	% Opt
CATS-regions	(100 problems)			
CPLEX	6.7	64117	0	100
Casanova$(.17, .5)$	4.2	1404	3.4	99.95
ESG$_h(1.9, .1, .01)$	7.2	1416	4.1	99.93
ASG$_h(.045, .01)$	12.7	2457	7.9	99.86
*ESG$_\ell(1.3, .9, .01)$	64.7	7948	77	88.11
*ASG$_\ell(.01, .01)$	48.1	9305	90	93.96
CATS-arbitrary	(100 problems)			
CPLEX	21.8	9510	0	100
Casanova$(.12, .04)$	8.7	2902	0.47	99.98
ESG$_h(2.1, .05, .5)$	32.5	7506	4.21	99.95
ASG$_h(.04, .01)$	30.2	6492	4.87	99.87
CATS-matching	(100 problems)			
CPLEX	1.30	499	0	100
Casanova$(.17, .3)$	.17	109	0	100
ESG$_h(1.7, .05, 0)$	.16	215	0	100
ASG$_h(.9, .8)$	.73	1248	0	100
CATS-paths	(100 problems)			
CPLEX	25	1	0	100
Casanova$(.15, .5)$	26	49	0	100
ESG$_h(1.3, .05, .4)$	28	2679	2.5	99.99
ASG$_h(.01, .15)$	75	5501	6.8	99.96
CATS-scheduling	(100 problems)			
CPLEX	15	1426	0	100
Casanova$(.04, .5)$	44	7017	19.9	99.87
ESG$_h(1.35, .05, .015)$	65	11737	41	99.68
ASG$_h(.015, .125)$	58	12925	44.2	99.51
Decay-200-200-.75	(100 problems)			
CPLEX	1.1	2014	0	100
Casanova$(.17, .5)$	0.5	2899	2	99.97
ESG$_h(14.5, .045, .45)$	1.8	24466	96.3	96.91
ASG$_h(.04, .5)$	1.7	24939	99.6	91.49
SAT(uf50)→CA	(10 problems)			
CPLEX	42	754	0	1
Casanova$(.11, .5)$	468	9.6×10^6	90	99.36
ESG$(30, .05, .1)$	31	1.7×10^6	10	99.95
ASG$(5, .5)$	173	8.6×10^6	80	99.40

Table 2: CA results

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+. 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 clearly indicate that these tend to be systematically easy problems for all the methods we tested, and therefore 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 envelop 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 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 2 gives our results for the default CATS problems, and shows that there is very little reason not to use CPLEX to solve these problems in practice.[4] 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. On these problems there seems to be little to choose, however, between additive and multiplicative updates. Table 2 shows that the hinge penalty is once again distinctly superior to the traditional linear penalty (as demonstrated by the results for ESG_ℓ and ASG_ℓ on the "regions" problems). On the more difficult SAT$\rightarrow$CA encoded problems (Table 2) we find that the local search methods still fail to show any significant improvement over CPLEX, but nevertheless all are challenged by these problems.

[4]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.

References

[Andersson *et al.*, 2000] A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings ICMAS-00*, pages 39–46, 2000.

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

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

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

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

[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. *Proceedings AAAI-00*, pages 22–29, 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.

[Leyton-Brown *et al.*, 2000] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auctions algorithms. *Proceedings 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.

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

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

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

[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.

[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.