Volume 8 Issue 2 May 2011
ISSN 1572-5286
ELSEVIER
DISCRETE OPTIMIZATION
Available online at www.sciencedirect.com
ScienceDirect

Discrete Optimization 8 (2011) 147–159

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

Information-theoretic approaches to branching in search[☆]

Andrew Gilpin, Tuomas Sandholm [*]

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA

ARTICLE INFO

Article history:
Received 17 September 2008
Received in revised form 29 December 2009
Accepted 7 July 2010
Available online 23 October 2010

Keywords:
Search
Branching heuristic
Branch-and-bound
Mixed-integer programming
Entropic branching

ABSTRACT

Deciding what question to branch on at each node is a key element of search algorithms. In this paper, we describe a collection of techniques for branching decisions that are motivated from an information-theoretic perspective. The idea is to drive the search to reduce the uncertainty (entropy) in the current subproblem. We present four families of methods for branch question selection in mixed integer programming that use this idea. In the first, a variable to branch on is selected based on lookahead. This method performs comparably to strong branching on MIPLIB, and better than strong branching on hard real-world procurement optimization instances on which CPLEX's default strong branching outperforms CPLEX's default branching strategy. The second family combines this idea with strong branching. The third family does not use lookahead, but instead exploits the tie between indicator variables and the variables they govern. This significantly outperforms the state-of-the-art branching strategies on both combinatorial procurement problems and facility location problems. The fourth family concerns branching using carefully constructed linear inequality constraints over *sets* of variables.

1. Introduction

Search is a fundamental technique for problem solving in artificial intelligence and operations research. At a node of the search tree, the search algorithm poses a question, and then tries out the different answers (which correspond to the branches emanating from that node). Many different ways of deciding which question to pose (branching strategies) have been studied.

In this paper we introduce a collection of techniques for branching decisions that are motivated from an information-theoretic perspective, employing information theory as a principle that guides the development. The strategies aim to reduce the uncertainty (entropy) in the current subtree. In the context of solving integer programs, we develop four high-level families of strategies that fall within this paradigm, and show that some of them have a significant improvement in speed over existing strategies. The rest of this section covers the background needed.

1.1. Mixed integer programming

One of the most important computational problems in operations research and computer science is integer programming. Applications of integer programming include scheduling, routing, VLSI circuit design, and facility location [1]. Integer programming is the problem of optimizing a linear function subject to linear constraints and integrality constraints on some of the variables. Formally, we have the following.

[☆] This was work was funded by, and conducted at, CombineNet, Inc., Fifteen 27th St., Pittsburgh, PA 15222. A short early version of this paper appeared at the International Joint Conference on Artificial Intelligence (IJCAI), 2007.

[*] Corresponding author. Tel.: +1 412 654 5994.
E-mail addresses: gilpin@cs.cmu.edu (A. Gilpin), sandholm@cs.cmu.edu (T. Sandholm).

1572-5286/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.disopt.2010.07.001

Definition 1 (*0–1 Integer Programming*).
Given an n-tuple c of rationals, an m-tuple b of rationals, and an $m \times n$ matrix A of rationals, the 0–1 *integer programming problem* is to find the n-tuple x such that $Ax \leq b$, $x \in \{0, 1\}^n$, and $c \cdot x$ is minimized.

If some variables are constrained to be integers (not necessarily binary), then the problem is simply called integer programming. If not all variables are constrained to be integral (they can be real), then the problem is called *mixed* integer programming (MIP). Otherwise, the problem is called *pure* integer programming.

While 0–1 integer programming (and thus integer programming and MIP as well) is $\mathcal{NP}$-hard [2], there are many sophisticated techniques that can solve very large instances in practice. We now briefly review the existing techniques upon which we build our methods.

1.1.1. Branch-and-bound

In *branch-and-bound* search, the best solution found so far (the *incumbent*) is kept in memory. Once a node in the search tree is generated, a lower bound (also known as an f-estimate) on the solution value is computed by solving a relaxed version of the problem, *while honoring the commitments made on the search path so far*. The most common method for doing this is to solve the problem while relaxing only the integrality constraints of all undecided variables; that *linear program* (LP) can be solved quickly in practice, for example using the simplex algorithm (and in polynomial worst-case time using interior-point methods). A path terminates when the lower bound is at least the value of the incumbent (or when the subproblem is infeasible or yields an integer solution). Once all paths have terminated, the incumbent is a provably optimal solution.

There are several ways to decide which leaf node of the search tree to expand next. For example, in *depth-first* branch-and-bound, the most recent node is expanded next. In *A^* search* [3], the leaf with the lowest lower bound is expanded next. A^* is desirable in the sense that, for any given branch question ordering, no tree search algorithm that finds a provably optimal solution can guarantee expanding fewer nodes [4]. Therefore, of the known node selection strategies, A^* seems best-suited when the goal is to find a provably optimal solution. An almost identical node selection strategy, *best-bound search*, is often used in MIP [5]. (The difference is that in A^* the children are evaluated when they are generated, while in best-bound search the children are queued for expansion based on their parents' values and the LP of each child is only solved if the child comes up for expansion from the queue. Thus best-bound search needs to continue until each node on the queue has value no better than the incumbent. Best-bound search generates more nodes, but may require fewer (or more) LPs to be solved.) Therefore, in the experiments we will use best-bound search in all of the algorithms.

1.1.2. Branch-and-cut

A more modern algorithm for solving MIPs is *branch-and-cut*, which first achieved success in solving large instances of the traveling salesman problem [6,7], and is now the core of the fastest commercial general-purpose integer programming packages. It is like branch-and-bound, except that, in addition, the algorithm may generate *cutting planes* [1]. They are linear constraints that, when added to the problem at a search node, result in a smaller feasible space for the LP (while not cutting off the optimal integer solution) and thus a higher lower bound. The higher lower bound in turn can cause earlier termination of the search path, and thus yields smaller search trees.

CPLEX [8] is a leading commercial software product for solving MIPs. It uses branch-and-cut. It can be configured to support many different branching algorithms, and it makes available low-level interfaces for controlling the search. We developed our branching methods in the framework of CPLEX, allowing us to take advantage of the many components already in CPLEX (such as the presolver, the cutting plane engine, the LP solver, primal heuristics, etc.) while allowing us flexibility in developing our own methods.[1]

Furthermore, the use of CPLEX allows us to study the impact of our techniques in the context of a full-blown state-of-the-art MIP solver rather than some simplification in which many of the techniques of modern MIP solvers might be missing or inefficiently implemented. Therefore, we obtain results that are meaningful from a practical perspective and are not overly optimistic because techniques in state-of-the-art MIP solvers that might partially substitute for the benefits of our techniques are properly captured.

We configured the search order to best-bound, and we varied the branching strategies, as we will discuss.

1.1.3. Selecting a question to branch on

At every node of a tree search, the search algorithm has to decide what question to branch on (and thus what the children of the node will be). The bulk of research has focused on branching on individual variables because that is intuitive, has a relatively small set of branching candidates, and tends to keep the LP sparse and thus relatively fast to solve. In other words, the question is: "What should the value of this variable be?". The children correspond to different answers to this question.

One commonly discussed method in operations research is to branch on the *most fractional variable*, i.e., variable whose LP value is furthest from being integral [5]. Finding a branching variable under this rule is fast, and the method yields

[1] We used CPLEX version 9.1, i.e., the newest version at the time of the experiments.

small search trees on many problem instances. (Recent research shows, however, that this method is not much better than randomly selecting a branching variable [9].)

A more sophisticated approach, which is better suited for certain hard problem instances, is *strong branching* [10]. The algorithm performs a one-step lookahead for each variable that is non-integral in the LP at the node. The one-step lookahead computations solve the LP relaxation for each of the children. (Often in practice, the lookahead is done only for *some* heuristically selected ones of those variables in order to reduce the number of LPs that need to be solved. Similarly, the child LPs are not solved to optimality; the amount of work (such as the number of simplex pivots) to perform on each child is a parameter. We will vary both of these parameters in our experiments.)

Algorithm 1 (*Strong Branching (SB) Variable Selection*)**.**

```
1. Let $x^*$ be an optimal solution to the current node's LP
2. candidates $\leftarrow \{i \mid x_i^* \text{fractional}\}$
3. For each $i \in$ candidates:
   (a) Let $z^l$ be the value of the LP under the additional constraint $x_i \leq \lfloor x_i^* \rfloor$
   (b) Let $z^u$ be the value of the LP under the additional constraint $x_i \geq \lceil x_i^* \rceil$
   (c) $\text{score}(x_i) \leftarrow 10 \cdot \min\{z^l, z^u\} + \max\{z^l, z^u\}$
4. $i^* \leftarrow \arg\max_{i \in \text{candidates}} \text{score}(x_i)$
5. Return $i^*$
```

There are many different ways that step 3(c) could be performed in Algorithm 1. The above method is from [10]. We experimented with the following variations in addition to the method above:

- $\text{score}(x_i) \leftarrow \min\{z^l, z^u\} + 10 \cdot \max\{z^l, z^u\}$
- $\text{score}(x_i) \leftarrow z^l + z^u$
- $\text{score}(x_i) \leftarrow \max\{z^l, z^u\}$
- $\text{score}(x_i) \leftarrow \min\{z^l, z^u\}$
- $\text{score}(x_i) \leftarrow (1 - x_i)z^l + x_i z^u$

In our preliminary experiments, there was no variation that dominated any of the others. We therefore decided to go with the variation in Algorithm 1, which has been shown to perform well in practice [10].

2. The information-theoretic approach to branching in search

The simple key observation behind our techniques is that, at the beginning of a search, the nodes on the frontier of the search tree have large amounts of uncertainty about the variables' values, while at the end of a search there is none (a path ends once there is no uncertainty left in a node's variable assignments). (In addition to all variables getting fixed value assignments, a path can end by infeasibility (the assignments so far implying a contradiction) or by fathoming by bound (the optimistic value of the node being no better than the value of the incumbent.)) Motivated by this observation, our techniques *guide the search so as to remove uncertainty from the nodes on the frontier of the search tree*. In this paper we apply this approach to decide what question should be branched on at each search node. While there has been much work on search (and specifically on developing branching heuristics), to our knowledge this is the first work that takes an information-theoretic approach to guiding the search process. (Another view of our approach is that we use information-theoretic measures to quantify how much propagation a candidate branching question would cause—not only counting how many of the unassigned variables would be affected, but also by how much.)

Specifically in the context of MIP, the idea we use is to treat the fractional portion of integer-constrained variables in the LP solution as probabilities, indicating the probability with which we expect the variable to be greater than its current value in the optimal solution. Clearly, interpreting LP variable values as independent probabilities is an enormous inaccurate assumption, and it is one that we approach with caution. Due to the constraints in the problem, the variables are indeed interdependent. However, because we are not attempting to derive theoretical results related to this assumption, and because we use the assumption only in deciding how to branch within a search framework (and thus we still guarantee optimality), this assumption does not negatively interfere with any of our results. As we demonstrate later in our experiments, this assumption works well in practice, so it is not without merit. We note that an in-depth study of the statistical relationship between fractional values and the optimal values is an interesting topic for further research. (Interpreting LP variables as probabilities is also used successfully in randomized rounding [11].)

Before we describe any of our specific families of branch question selection, it will be useful to define how we can quantify the "uncertainty" of a partial solution. For this, we borrow some definitions from information theory, from which the primary contribution is the notion of *entropy* [12], which measures the amount of uncertainty in a random event. Given an event with two outcomes (say 0 or 1), we can compute the entropy of the event from the probability of each outcome occurring.

Definition 2 (*Entropy of a Binary Variable*).
Consider an event with two outcomes, 0 and 1. Let x be the probability of outcome 1 occurring. Then $1-x$ is the probability of outcome 0 occurring, and we can compute the entropy of x as follows:

$$e(x) = \begin{cases} -x\log_2 x - (1-x)\log_2(1-x) & 0 < x < 1 \\ 0 & x \in \{0, 1\}. \end{cases}$$

We define $e(0) = e(1) = 0$ since $\log_2 0$ is undefined.

It is possible to use other functions to measure the uncertainty in a binary variable. For example,

$$e(x) = \frac{1}{2} - \left|\frac{1}{2} - x\right|$$

and

$$e(x) = x - x^2$$

could alternatively be used. In the context of the first family of branching question selection techniques (discussed below), we experimented using these other functions and observed very similar results as compared to using $e(x)$ as in Definition 2. (This is not too surprising in light of the fact that the three measures have very similar numerical behavior.) Thus, throughout the rest of the paper, we will use this standard way of calculating entropy.

Entropy is additive for independent variables, so we compute the entropy of a group of variables as follows.

Definition 3 (*Entropy of a Group of Binary Variables*).
Given a set $\mathcal{X}$ of probabilities corresponding to independent binary events, we can compute the entropy of the set as

$$\text{entropy}\,(\mathcal{X}) = \sum_{x\in\mathcal{X}} e(x),$$

where $e(x)$ is as in Definition 2.

While it is possible for there to be multiple optimal solutions to an optimization problem, all optimal solutions (and, in fact, all feasible solutions) will have zero uncertainty according to this measure.

When a variable x is constrained to be integer (not necessarily binary), we measure $e(\cdot)$ on the fractional portion of the variable: $x - \lfloor x \rfloor$.

We are now ready to present our four families of branching question selection techniques. They all fall under the information-theoretic approach for branching.

3. Family 1: entropic lookahead for variable selection

In the first family, we determine the variable to branch on using one-step lookahead as in strong branching. The difference is that, instead of examining the objective values of potential child nodes, we examine the remaining uncertainty (entropy) in potential child nodes, choosing a variable to branch on that yields children with the least uncertainty.

Algorithm 2 (*Entropic Branching (EB)*).

```
1. candidates $\leftarrow \{i \mid x_i \text{ fractional}\}$
2. For each $i \in$ candidates:
   (a) $x_f \leftarrow x_i - \lfloor x_i \rfloor$
   (b) Let $\hat{x}^l$ be solution vector of LP with $\hat{x}^l_i \le \lfloor x_i \rfloor$
   (c) Let $\hat{x}^u$ be solution vector of LP with $\hat{x}^u_i \ge \lceil x_i \rceil$
   (d) entropy$(x_i) \leftarrow \sum_{j=1}^{n} (1 - x_f)e\left(\hat{x}^l_j\right) + x_f e\left(\hat{x}^u_j\right)$
3. $i^* \leftarrow \arg\min_{i\in\text{candidates}}$ entropy$(x_i)$
4. Return $i^*$
```

EB is usable on any MIP since it does not make any assumptions about the underlying model for the problem on which it is used.

EB (and SB) can be modified to perform more than one-step lookahead in the obvious way. This would tend to lead to smaller search trees but more time would be spent per node. (For SB, two-step lookahead has been shown to pay off in overall run-time on certain very hard problems (when additionally using implication information obtained from the two-step lookahead) [13].) One way of mitigating this tradeoff would be to conduct deeper lookaheads only on candidates that look promising based on shallower lookaheads. One could even curtail the candidate set based on heuristics before any lookahead is conducted.

For illustrative purposes, there is an interesting (though not exact) analogy between our entropic lookahead method for question selection at search nodes and algorithms for decision tree induction [14]. In most recursive decision tree induction algorithms, a question is inserted at a leaf that results in the greatest information gain. Similarly, in search, by choosing questions whose children have the least entropy, we are creating children that in a sense also result in the greatest information gain.

4. Family 2: hybridizing SB and EB

As can be observed from the pseudocode given in Algorithms 1 and 2, SB and EB are computed quite similarly. A natural question arises: can we develop a hybrid approach combining the strengths of both without using significantly more computational resources? In this section we answer this question in the affirmative by introducing a second family of variable selection strategies. In this family, SB and EB are hybridized in different ways. Each of these methods requires only a small amount of additional computation compared to performing *only* SB or EB (because the same lookahead with the same child LP solves is used). We classify our hybrid approaches into two categories: tie-breaking and combinational.

4.1. Tie-breaking methods

In this approach, we first perform the SB computations as in Algorithm 1, but instead of simply branching on the variable with best score, we break ties using an entropy computation. Since we have already computed the LP relaxations for each branch, computing the entropy is a negligible computational cost (relative to computing the LP relaxations). In addition to breaking exact ties, we also experimented with the approach of considering two variables having SB scores within $p\%$ of each other as tied. In the experiments (described below) we tested with $p \in \{0, 5, 10\}$. For a given percentage p, we refer to this method as TB($p\%$).

4.2. Combinational methods

We present two methods of combining information from SB and EB in order to compute a single score for a variable. The variable with the best score will then be branched on.

The first method, RANK, performs the computation for SB and EB first. (Again, these computations are performed simultaneously at little additional cost beyond doing either SB or EB alone.) Define $\text{rank}_{\text{SB}}(x_i)$ to be the rank of variable x_i in terms of its SB score (i.e., the variable with the largest score would have rank 1, the variable with the second-largest score would have rank 2, and so on). Similarly, define $\text{rank}_{\text{EB}}(x_i)$ to be the rank of variable x_i in terms of its EB entropy. Then, for each variable, we let $\text{rank}(x_i) = \text{rank}_{\text{SB}}(x_i) + \text{rank}_{\text{EB}}(x_i)$ and choose the variable x_i with the smallest rank.

The second method, COMB(ρ, $1 - \rho$), computes a convex combination of the SB score (with weight ρ) and the current entropy minus the EB score (with weight $1 - \rho$). It then selects the variable with the highest final score.

4.3. Experiments on EB and the hybrid methods

We conducted a host of experiments with the search methods described above, both on MIPLIB and real-world procurement optimization instances. In all of our experiments the algorithms ran in main memory, so paging was not an issue.

In order to be able to carefully and fairly control the parameter settings of both SB and EB, we implemented both. (Using CPLEX's default SB would not have allowed us to control the proprietary and undocumented candidate selection method and scoring function, and CPLEX does not provide adequate APIs to use those same methods for EB.) Implementation of both algorithms in the same codebase also minimizes differences in implementation-related run-time overhead.

4.3.1. Experiments on MIPLIB 3.0

MIPLIB [15] is a library of MIP instances that is commonly used for benchmarking MIP algorithms.

We experimented on all instances of MIPLIB 3.0. The detailed results are in the Appendix; we summarize the key results here.

There is no clear winner between EB and SB. EB reached the 1 h time limit on 34.7% of the instances, while SB timed out on 24.5% of the instances. EB outperformed SB on 13 of the 49 instances. We find this interesting in light of the fact that EB does not use the objective function of the problem at all in making branching decisions.

Our experiments show that, on average, RANK is the best among the hybrid methods. On 17 of the instances, at least one of the hybrid methods outperformed both SB and EB.

Although EB performs comparably to SB, both of these strategies are dominated on MIPLIB by CPLEX's default branching strategy (although it searches a larger number of nodes). For problems with lots of structure, we expect the reverse to be true. Indeed, in the next subsection we use instances on which CPLEX's default SB outperforms CPLEX's default branching strategy, and we demonstrate that EB outperforms SB on that data set.

4.3.2. Experiments on real-world procurement optimization

As we showed in the previous section, lookahead (such as in strong branching) does not pay off on all classes of problems. To obtain a pertinent evaluation of EB against the state-of-the-art lookahead-based technique, SB, we wanted to test on a problem set on which SB is the algorithm of choice (compared to non-lookahead-based standard techniques). We ran experiments on CombineNet, Inc.'s repository of thousands of large-scale real-world industrial procurement optimization instances of varying sizes and structures, and selected the instances on which CPLEX's default implementation of SB was

Table 1
Average computation time (in seconds) over the 121 instances for entropic branching with a 1 h time limit.

Candidates	10 iterations	25 iterations	100 iterations	Unlimited iterations
10	1947.00	2051.07	1966.21	1913.56
Unlimited	1916.81	1962.09	1813.76	1696.53

Table 2
Median computation time (in seconds) over the 121 instances for entropic branching with a 1 h time limit.

Candidates	10 iterations	25 iterations	100 iterations	Unlimited iterations
10	3600	3600	3600	3600
Unlimited	2955.78	3195.58	1374.679	590.23

Table 3
Number of instances (of the 121) taking over an hour.

Candidates	10 iterations	25 iterations	100 iterations	Unlimited iterations
10	62	65	62	61
Unlimited	59	57	54	49

faster than CPLEX's default branching strategy. There were 121 instances like that. Those instances had an average of 59,303 variables and 67,673 constraints. On those instances, CPLEX's default implementation of SB was on average 27% faster than CPLEX's default branching strategy. Thus we concluded that SB is a good algorithm for that data set, and performed a comparison of EB against SB on that data.

Tables 1–3 summarize the experimental results for EB. We varied the size of the candidate list and the number of simplex iterations performed in the dual LP of each child of the candidate. (As usual in MIP, we tackle the dual LP instead of the primal LP because a node's dual solution serves as a feasible basis for the child and thus serves as a hot start for simplex. The first child's dual LP solve starts from the parent's dual LP basis. The second child's LP solve starts from the first child's LP basis, which we experimentally observed was faster than starting from the parent's basis.) When limiting the size of the candidate list, we choose the candidates in order of most fractional.

As the tables demonstrate, EB was fastest with the most detailed branch selection. The additional work in performing more simplex iterations pays off by giving a more accurate estimate of the entropy of the individual variables. Similarly, by examining more candidate variables, EB is more likely to branch on a variable that decreases the total amount of entropy the most. This suggests that a better method for choosing the candidate list could lead to further speedup.

When both methods were allowed unlimited candidates and dual iterations, EB was 29.5% faster than SB: the comparable number to EB's 1696.53 average seconds was 2406.07 for SB.

5. Family 3: entropic lookahead-free variable selection

In this section we introduce a third family of branching strategies, again within the entropy-based branching approach. This method is computationally less expensive than the methods we presented so far because it does not use lookahead. However, it does require some knowledge of the structure of the problem.

We examine two different classes of problems. In Section 5.1 we study combinatorial procurement auctions, and in Section 5.2 we study facility location problems.

5.1. Combinatorial procurement auctions

The first problem with which we experiment is motivated by a real-world electronic commerce application: a combinatorial procurement auction (also known as reverse auction) where the buyer specifies the maximum number of winning suppliers [16,17].

Definition 4 (*Combinatorial Procurement Auction with Maximum Winners Constraint*)**.**
Let $\mathcal{M} = \{1, \ldots, m\}$ be the m goods that the buyer wishes to procure (the buyer wants at least one unit of each good). Let $\mathcal{S} = \{1, \ldots, s\}$ be the s suppliers participating in the procurement and let $\mathcal{B} = \{B_1, \ldots, B_n\}$ be the bids, where $B_i = \langle G_i, s_i, p_i \rangle$ indicates that supplier s_i can supply the bundle of goods $G_i \subseteq \mathcal{M}$ at price p_i. Finally, the buyer indicates the maximum number of winners, k. The winner determination problem is to identify the winning bids so as to minimize the buyer's cost subject to the constraints that the buyer's demand is satisfied and that the maximum number of winners constraint is satisfied.

This problem is $\mathcal{NP}$-complete, even if the bids are on single items only [17]. Integer programming methods have been successfully used previously in winner determination research (see, e.g., [18–23]), and we can very naturally formulate the above generalized winner determination problem as an MIP:

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{i=1}^{n} p_i x_i & \\
\text{such that} & \displaystyle\sum_{i \mid j \in G_i} x_i \geq 1 & j \in \{1, \ldots, m\} \\
& \displaystyle\sum_{i \mid s_i = j} x_i - m y_j \leq 0 & j \in \{1, \ldots, s\} \\
& \displaystyle\sum_{j=1}^{s} y_j - k \leq 0 & \\
& x_i \in \{0, 1\} & i \in \{1, \ldots, n\} \\
& y_j \in \{0, 1\} & j \in \{1, \ldots, s\}.
\end{array}$$

The formulation is typical of a common class of problems in which binary "indicator" variables—the y_j variables in the formulation above—are used to model logical connectives [1]. Constraints that state that at most (or exactly) k variables from a set of variables can be nonzero are an important special case. (The case $k = 1$ is called a *special-ordered set of Type I* [24].) Typically, the LP relaxation gives poor approximate values for indicator variables: due to big M's in the constraints that include the indicators (and even with large m's that are as small as possible as in the above formulation), an indicator can in effect be on even while taking a tiny value (or conversely, be off while holding a value close to 1). As we show, the branching method that we propose helps significantly to address this problem.

5.1.1. *Branching strategy*

The hardness in this problem comes primarily from determining the winning set of suppliers. In terms of the above MIP, we need to determine the y_j values. The main idea in our branching strategy for this problem is to branch on y_j values that correspond to suppliers about which we are most uncertain. But rather than deriving this uncertainty from the variable y_j (for which the LP gives very inaccurate values), we derive it from the variables corresponding to the bids of supplier j. The branching strategy works as follows. For each supplier j, where $y_j \notin \{0, 1\}$, compute

$$\text{entropy}(j) = \sum_{i \mid s_i = j} e(x_i)$$

and branch on the variable y_{j^*}, where

$$j^* = \arg\max_j \text{entropy}(j).$$

This strategy does not use lookahead: it only uses the LP values of the current search node. We call this branching strategy *indicator entropic branching* (IEB).

5.1.2. *Experimental results*

Although this problem is motivated by a real-world application, there are no publicly available real-world instances (the real-world data studied in the previous section does not exactly fit this model). Furthermore, none of the combinatorial auction instance generators published in the literature have a notion of supplier. Instead we created an artificial instance distribution for this problem which closely approximates the real-world problem.

Given parameters s (number of suppliers), r (number of regions), m (goods per region), and b (bids per region), we create an instance of a procurement auction with a maximum winners constraint as follows.

1. Each bidder bids on a region with probability 0.9.
2. For each region, generate the bidder's bids using the Decay distribution [25] with $\alpha = 0.75$. Each bid in the Decay distribution is generated as follows. Give the bid one random item from $\mathcal{M}$. Then repeatedly add a new random item from $\mathcal{M}$ (without replacement) with probability α until an item is not added or the bid includes all m items. Pick the price uniformly between 0 and the number of items in the bid.

For this data distribution, we determined that CPLEX's default branching strategy was the best of the four qualitatively different branching strategies that CPLEX offers. In particular, it was faster than strong branching on this data. Table 4 shows experimental results comparing IEB with CPLEX using its default branching strategy. The results indicate that IEB performs significantly better, and the relative difference increases with problem size. (We also found that part of the benefit can be achieved even without entropic branching by simply forcing branching on every path to occur on the fractional indicator variables first before branching on any other variables. Other than that, we let CPLEX make the variable selection in that strategy (CPLEX-IF)).

5.2. *Facility location*

In the uncapacitated facility location problem, there are n facilities and m customers. The cost to open facility j is c_j, and the cost of servicing customer i using facility i is c_{ji}. The goal is to minimize the overall cost of opening facilities and servicing

Table 4
The first two rows contain the solution time (in seconds) for finding the optimal solution and proving optimality averaged over 25 instances (there were no timeouts). The third row indicates the average integrality gap (how far from optimal (at worst) the best solution found so far is) after one hour.

Suppliers s	Regions r	Goods per region m	Bids per region b	Max winners k	CPLEX	CPLEX-IF	IEB
20	10	10	100	5	25.63	15.13	11.81
30	15	15	150	8	5755.92	684.83	551.82
40	20	20	200	10	37.05%	32.38%	30.57%

customers such that every customer is served. In the MIP formulation below, y_j is the decision variable for opening facility j, and x_{ji} is the decision variable for servicing customer i with facility j.

$$\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{j=1}^{n} c_j y_j + \sum_{j=1}^{n}\sum_{i=1}^{m} c_{ji} x_{ji} & \\
\text{such that} & x_{ji} \le y_j & \text{for all } i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\} \\
 & \displaystyle\sum_{j=1}^{n} x_{ji} \ge 1 & \text{for all } i \in \{1,\ldots,m\} \\
 & x_{ji} \in \{0,1\} & \text{for all } i \in \{1,\ldots,m\}, j \in \{1,\ldots,n\} \\
 & y_j \in \{0,1\} & \text{for all } j \in \{1,\ldots,n\}.
\end{array}$$

5.2.1. *Branching strategy*

As in the combinatorial procurement problem, the difficulty in solving facility location problems comes primarily from determining which facilities to open. In terms of the above MIP, this corresponds to determining the values for the y_j variables. Similarly to our branching strategy for combinatorial procurement problems, we branch on y_j values that correspond to customers about which we are most uncertain. For each facility j, where $y_j \notin \{0, 1\}$, we compute

$$\text{entropy}(j) = \sum_{i=1}^{m} e(x_{ji})$$

and branch on the variable y_{j^*}, where

$$j^* = \text{argmax}_j \text{ entropy}(j).$$

Given how similar this branching strategy is to our strategy for combinatorial procurement problems, we also call this branching strategy *indicator entropic branching* (IEB).

5.2.2. *Experiments*

To evaluate the branching heuristics, we used instances from the Uflib problem repository.[2] This collection of instances contains a number of families of problems. We evaluated the heuristics on all of the problems in that repository. For the Chess, Euclid, and GalvaoRaggi problem families, CPLEX was able to solve the problem optimally at the root node through a combination of presolve and cutting planes. Thus, we do not present any results for those problems.

Table 5 contains our results for the four Bilde–Krarup families [26], the Finite projective plane instances (Fpp11 and Fpp17) [27], and the Uniform instances [27]. In all cases, indicator entropic branching (IEB) uses fewer search nodes than branching on indicators first (CPLEX-IF) and the default CPLEX branching strategy. Overall, IEB is 2.18 times faster than the default CPLEX branching strategy and 1.48 times faster than branching on indicators first. In terms of the number of nodes searched, IEB uses 93% fewer search nodes than default CPLEX and 68% fewer search nodes than branching on indicators first.

6. Family 4: entropic lookahead for multi-variable branches

In this section, we introduce a fourth family of methods for determining a good question to branch on. In EB, we performed a one-step lookahead for each non-integral variable. Here, we generalize entropic lookahead beyond branching on variables. In integer programming one can branch on the sum of the values of a *set* of variables. For example, if the LP relaxation at the current search node has $x_i = 0.2$ and $x_j = 0.6$, we could set one branch to be $x_i + x_j \le 0$ and the other branch to be $x_i + x_j \ge 1$. In general, given a set $\mathcal{X}$ of variables and the current LP relaxation solution $\hat{x}$, we can let

$$k = \left\lfloor \sum_{i \in \mathcal{X}} \hat{x}_i \right\rfloor, \tag{1}$$

and we can generate the branches

[2] http://www.mpi-inf.mpg.de/departments/d1/projects/benchmarks/UflLib/index.html.

Table 5
Experimental results comparing the various branching algorithms on the Uflib problem instances. For each algorithm, the table shows the average solve time (in seconds) and the average number of nodes searched.

Instance family	Instance Info			CPLEX		CPLEX-IF		IEB	
	n	m	#Instances	Nodes	Time	Nodes	Time	Nodes	Time
BildeKrarup-B	30	80	10	10.2	2.6	7.6	2.8	4.7	2.7
BildeKrarup-C	50	100	10	2467.2	81.0	377.2	40.7	143.0	29.4
BildeKrarup-D	30	80	100	554.9	16.0	88.4	10.5	46.2	8.5
BildeKrarup-E	50	100	100	2983.9	162.0	383.9	65.7	145.8	47.3
Fpp-11	133	133	30	2.5	5.15	3.0	8.65	1.3	5.16
Fpp-17	307	307	30	1.5	99.1	2.3	163.0	0.8	95.5
Uniform	100	100	30	3513.3	186.1	1793.0	207.0	421.8	133.5

$$\sum_{i\in\mathcal{X}} x_i \le k \quad \text{and} \quad \sum_{i\in\mathcal{X}} x_i \ge k+1. \tag{2}$$

No other value of k is worth considering: any other integer value would cause one child's LP optimum to be exactly the same as the node's. Thus that child's EB analysis (or SB analysis, or in fact any branching rule based solely on the node's local information) would be the same as that of the node's, leading to an infinitely deep search path and non-termination of the overall algorithm.

Now, instead of branching on the variable with the smallest amount of entropy in its child nodes, we select the *set* $\mathcal{X}$ of variables for branching that results in the smallest amount of entropy in the two child nodes. (We never branch on a set of variables $\mathcal{X}$ if $\sum_{i\in\mathcal{X}} \hat{x}_i$ happens to be integral because one of the branches will not constrain the current LP solution. Thus the corresponding child node will be identical to its parent, leading again to non-termination with any branching rule that is based on local information. In fact, we do not even conduct the lookahead for such $\mathcal{X}$.) In step 2(d) of EB, we weighted the entropy of each child by the probability that we expect the optimal solution to occur in each child. In the multi-variable case, we still perform this weighting, but it is more complicated since the probability of each branch depends on several variables. The probability that the sum is less than k is the summation of a combinatorial number of products of probabilities; this number is exponential only in $|\mathcal{X}|$, so it is not prohibitive for generating branches with small numbers of variables.

More generally, one can branch on the disjunction

$$\sum_{i\in\mathcal{X}} a_i x_i \le r$$

versus

$$\sum_{i\in\mathcal{X}} a_i x_i \ge r+1,$$

where the constants a_i can be positive or negative, as long as there are no integer solutions between those two hyperplanes (see, e.g., [28,29]). Our entropy reduction measure could then be used to select from among such pairs of hyperplanes. Branching on high-level properties has also been shown efficient in constraint satisfaction problems [30], and it would be interesting to try the information-theoretic approach for branch question selection in that context as well.

While branching on more than one variable at a time may seem unintuitive, it does not cause any obvious loss in branching power.

Proposition 1. *Assume that each integer variable x_i has finite domain D_i. Ignoring the effects of pruning (by infeasibility, bound, and LP integrality), propagation (by LP), and learning (for example, from conflicts in subproblems), the number of leaves in the tree is the same regardless of how many (and which) variables are used in different branches—as long as trivial branches where a child is identical to its parent are not used. If the branching factor is a constant, then this equality applies to the number of nodes in the tree as well.*

Proof. There are a finite number, K, of feasible solutions. (K is at most $\prod_{i=1}^{n} |D_i|$, but may be less than that because there may be additional constraints in the model beyond the variable bounds and the integrality constraints.) Regardless of the branching rule, the tree will have one leaf corresponding to each of the K solutions. Therefore, the number of leaves is the same across the trees. Finally, constant branching factor trees with the same number of leaves have the same number of nodes. □

We performed experiments on MIPLIB 3.0 and on combinatorial auction winner determination problem instances (again from the Decay distribution, with several values of α) using the branching questions of Eq. (2) as the branching candidates. We limited our algorithm to considering candidates containing just one or two variables in order to keep the number of candidates small so as to not have to solve too many child LPs. This also helps keep the LP sparse and thus relatively fast to solve.

While this strategy led to trees that are slightly smaller on average than when branching on individual variables only, the computational effort needed to perform the lookahead for pairs of variables was not worth it in terms of total search time. It thus seems that, in order for this method to be effective, there needs to be some quick way of determining (before lookahead) what good candidate variable sets to branch on might be, and to only conduct the lookahead on them. We also tried randomly picking only a restricted number of variable pairs as candidates; even though that helped in overall run-time, it did not help enough to beat branching on individual variables only. Hopefully future research will shed light on what might be considered good multi-variable branching candidates.

7. Discussion

Our current view is that there are now three known high-level approaches to branch question selection in search, as we will list below. (With the depth-first node selection strategy, an additional effective method is to branch on a question for which the algorithm knows the right answer with high confidence, and then proceed to exploring the child corresponding to the "better" variable assignment first (see, e.g., [19,20]). The motivation is that this leads to good feasible solutions being found early, so hopefully many of the "worse" children never need to be explored because they get fathomed.)

1. Branching on the question that tightens the node's lower bound the most. The motivation is that at the end of the search the lower bound has to reach or exceed the optimum on every path—thus branching to tighten the lower bound is a good way to branch. *Strong branching*, discussed above, is one example. Another example is *pseudocost branching* [31–33]: the search records for each variable that is branched on how much the lower bound tightened; at later nodes of the search tree those numbers are then used to branch on the variable that tightened the lower bound the most. Another improvement is to use strong branching for the first few levels of the tree and then use pseudocost branching [34]. This mitigates the initialization problem of pseudocost branching because the pseudocosts can be initialized based on the strong branching. Another method is to use strong branching only when the pseudocosts are uninitialized [33]. Yet another method is *reliability branching* [9], where pseudocosts are used if enough samples for a variable have been collected, and strong branching is used otherwise.
2. Branching on big questions first (closest to the root of the tree). This idea is embraced by many MIP practitioners. The motivation is that branching on big questions will cause significant propagation, for example, by the LP, and thus less branching is required overall. (It tends to also tighten the lower bound more than branching on smaller questions.) Another view is that big questions lead to child nodes that are very different from each other, so hopefully one of them will be noncompetitive and thus will be fathomed during the search.
   Usually, big questions are identified by the modeler using domain-specific exogenous knowledge. Branching on indicator variables first is an example of a domain-independent approach to branching on big questions first. Another method that falls under the idea of branching on big questions first is the balanced way of branching on special ordered sets (of Type I or II) [24]. The recent approach of branching on a variable that is involved in the largest number of active constraints [35] is also an example of this approach. The motivation was that branching on such a variable tends to yield children whose LP variable assignments are very different from each other and tends to cause a large amount of propagation. Experiments show that method to be efficient at finding good feasible solutions. We view that method as an automated domain-independent way of identifying a big question to branch on.
3. Our information-theoretic approach of branching on the question that reduces uncertainty in the node's variables the most. The motivation is that, at the beginning of a search, the nodes on the frontier of the search tree have large amounts of uncertainty about the variables' values, while at the end of a search there is none (a path ends once there is no uncertainty left in a node's variable assignments). Therefore, our approach is to guide the search so as to remove uncertainty from the nodes on the frontier.
   (Another view of our approach is that we use information-theoretic measures to quantify how much propagation a candidate branching question would cause—not only counting how many of the unassigned variables would be affected, but also by how much. In this sense our approach is related to approach (2) above.)
   One can view many existing search methods as quick approximations of our entropy-based search. First, the classic operations research idea of branching on the variable that has the most fractional LP value at the node in best-bound (A^*) search [5] is a lookahead-free approximation of entropy-based variable selection: it also tries to branch on the variable that the LP is most uncertain about and thus that branch should reduce uncertainty the most. Second, using entropy as the f-function in A^* search (i.e., always picking the node with least entropy to expand next) tends to make A^* more like *depth-first search* because deeper nodes tend to have less entropy. (However, entropy does not always decrease monotonically even along one search path.) Third, the most common heuristic in constraint satisfaction problems, *most-constrained-variable-first* [36], and its usual tie-breaker, the *most-constraining-variable-first* heuristic [37], approximate entropy-based variable selection: they tend to assign a variable that affects the other unassigned variables the most, thus reducing the entropy of those variables.

One can understand the existence of these three high-level approaches by interpreting the goal of the branch question selection to be that of keeping the search paths short. There are three ways in which a path can end in MIP: by fathoming, by infeasibility, or by integrality. Approach (1) above is geared toward achieving early fathoming, approach (2) toward early infeasibility (or feasible solution), and approach (3) toward early integrality. Because approach (3) tries to reduce uncertainty as quickly as possible, it also leads to early infeasibility (or feasible solution) and early exact node evaluation (leading to early fathoming).

Besides approach (3) above, one could try to branch toward integrality with other methods. For example, one could try to branch in a way that causes the LP table to get closer to satisfying some of the known sufficient conditions for the LP to have only integral vertices (for example, total unimodularity of the left-hand matrix A with integral right-hand side b). In this vein, the special case $k = 0$ of the branching question in Eq. (2) has been shown to be effective in set partitioning, set packing, and set covering problems [38–40]. For such 0–1 problems, a theory has been developed that shows that each path consisting of carefully selected branches will yield an LP with all vertices integral after a quadratic number of branches in the number of constraints [41]. Future research includes exploring the use of techniques from our Family 4 (Section 6) to select among such branching candidates in those problems. Besides branching toward LP integrality, one can branch so as to drive the search toward the remaining problem being solvable in polynomial time using special-purpose algorithms [42,19,20].

8. Conclusions

We have described a new technique for search guidance based on an information-theoretic approach. The simple key observation is that, at the beginning of search, the nodes on the frontier of the search tree have large amounts of uncertainty about the variables' values, while at the end of a search there is none (a path ends once there is no uncertainty left in a node's variable assignments). Motivated by this observation, our branching heuristics *guide the search so as to remove uncertainty from the nodes on the frontier of the search tree*.

In this paper, we applied this idea to decide what question should be branched on at each search node. While there has been much work on search (and specifically on developing branching heuristics), to our knowledge this is the first work that takes an information-theoretic approach to guiding the search process. Two high-level approaches to branching have been studied in the past: branching to tighten the lower bound and branching on big questions first. Our principle of uncertainty removal at the fringe nodes is a third high-level approach.

Specifically in the context of MIP, we treat the fractional portion of integer-constrained variables in the LP solution as probabilities, indicating the probability with which we expect the variable to be greater than its current value in the optimal solution. We then use entropy of the LP variables as the measure of uncertainty at each node.

Using this approach, we developed four families of methods for selecting what question to branch on. All four families are information-theoretically motivated to reduce remaining entropy. In the first family, a good variable to branch on is selected based on lookahead. Experiments show that this entropic branching method performs comparably to strong branching (a classic technique that uses lookahead and LP bounds to guide the search) on MIPLIB, and better than strong branching on hard real-world procurement optimization instances on which CPLEX's default strong branching outperforms CPLEX's default branching strategy. The second family combines this idea with strong branching in different ways. The third family does not use lookahead, but instead exploits the tie between indicator variables and the other variables they govern. Experiments show that this family significantly outperforms the state-of-the-art branching strategies on both combinatorial procurement problems and facility location problems. The fourth family concerns branching using carefully constructed linear inequality constraints over *sets* of variables.

9. Future research

There are several promising directions for future research down this path.

One could develop quick heuristics to curtail the set of candidate questions to branch on before lookahead. One could even narrow down the candidate set incrementally by deeper and deeper lookahead (the leaves being evaluated using our entropic measure). These issues should be tackled both for branching on individual variables and for branching on disjunctions involving multiple variables.

It would also be interesting to develop entropy-based branching techniques on additional problems, for example, on set covering, packing, and partitioning problems—perhaps using the specialized branching constraints (discussed above) as the candidates.

We explored hybrids between our high-level approach and the high-level approach of branching to tighten the lower bound (specifically, strong branching). Other hybrids are also possible. For example, one can explore hybrids between our branching approach and the second high-level branching approach (branching on big questions first). We developed a form of this in the IEB strategy which branches on indicators first, but selects among them using an entropic approach. Another interesting avenue in this same intersection would be to hybridize our approach with the idea of branching on questions that are big in the sense that the children are very different from each other in terms of the variable assignments in their LPs. The latter measure of "bigness" was the motivation behind Patel and Chinneck's approach [35]. (This criterion and ours are orthogonal. For example, both children of a node could have integral and very similar LP vectors. Such a branching question would score highly in our criterion but miserably in any criterion that attempts to generate children whose LP values are very different.) In practice Patel and Chinneck avoid solving the child LPs by instead branching on the variable that is involved in the largest number of active constraints. We could avoid that approximation, and use the actual difference of the children's LP vectors instead, because EB solves the child LPs anyway (at least approximately).

Our methods were largely derived for A^* (best-bound) search where it is best to branch on a question about which the algorithm is most uncertain; in contrast, in the depth-first node selection strategy it is often best to branch on a question for

Table 6
Number of nodes searched for different branching strategies. This is for finding the optimal solution and proving optimality, except where a percentage is given. In this case the table entry indicates the gap when the 1 h limit was reached.

Problem	SB	EB	TB (0%)	TB (5%)	TB (10%)	RANK
10teams	167	**16**	36	25	20	140
air03	2	2	2	2	2	2
air04	77	**25**	0.241%	1.29%	1.05%	1.14%
air05	4.74%	4.99%	1.32%	2.69%	7.42%	**1.06**%
cap6000	**4714**	0.000629%	6528	0.000642%	0.000642%	5460
dano3mip	21.39%	21.39%	21.39%	21.39%	21.39%	21.39%
danoint	**3.70**%	4.30%	3.76%	4.23%	4.29%	4.30%
dcmulti	**72**	152	96	2388	3886	86
egout	2	**0**	2	**0**	**0**	**0**
enigma	678	4276	1356	**310**	**310**	1371
fast0507	**6.33**%	6.43%	**6.33**%	6.44%	6.43%	6.41%
fiber	28	178	26	**24**	26	104
fixnet6	**54**	3074	**54**	102	238	198
harp2	**0.122**%	0.637%	0.477%	0.637%	0.637%	0.537%
khb05250	6	**4**	6	6	6	**4**
l152lav	**184**	0.412%	186	722	4296	1369
lseu	60	74	60	60	72	**52**
markshare1	100%	100%	100%	100%	100%	100%
markshare2	100%	100%	100%	100%	100%	100%
mas74	**2.11**%	7.69%	2.13%	3.95%	4.78%	5.29%
mas76	**444 916**	1.33%	**444 916**	0.342%	0.792%	0.864%
misc03	311	318	367	363	375	**260**
misc06	**19**	54	**19**	54	54	25
misc07	21 551	15 508	13 370	12 537	**12 424**	18 812
mitre	60	24	53	68	68	**5**
mkc	**1.95**%	10.4%	13.5%	11.6%	11.6%	13.3%
mod008	238	978	**234**	346	442	426
mod010	**10**	108	16	57	90	12
mod011	**33**	7.44%	11.5%	11.5%	11.5%	11.5%
modglob	**55**	306	56	286	308	126
nw04	**136**	765	294	387	610	165
p0033	4	**2**	4	4	4	**2**
p0201	94	172	230	490	650	**62**
p0282	10	10	40	16	232	**4**
p0548	0	0	0	0	0	0
p2756	**24**	36	**24**	70	35	30
pk1	267 226	37.3%	253 939	261 168	**246 642**	14.1%
pp08a	**314**	1298	441	1174	875	394
pp08aCUTS	**934**	8138	993	6207	7419	1652
qiu	**311.9**%	2400.8%	**311.9**%	1646.5%	1646.5%	357.6%
rentacar	**6**	10	**6**	**6**	**6**	10
rgn	2114	2608	2198	2070	1936	**1774**
set1ch	158	3228	**154**	294	714	414
seymour	**6.19**%	6.47%	**6.19**%	6.49%	6.43%	6.51%
stein27	2214	1472	2938	1664	1668	**1320**
stein45	26 194	22 454	46 032	30 544	30 302	**17 220**
swath	67.6%	72.5%	71.6%	**64.0**%	65.0%	75.4%
vpm1	0	0	0	0	0	0
vpm2	3350	2696	**1235**	1869	3478	2374

which the algorithm knows the right answer with high confidence [19,20]. Thus it is not clear that these branch question selection techniques would work as well under the depth-first strategy. That is a future question to pursue. Our approach should also be tested in settings where the goal is not provable optimality but finding a good feasible solution quickly. One could also develop entropy-based branching techniques on constraint satisfaction problems; given that there are no LP values, one would need another way of measuring entropy.

While we introduced four families of branch question selection techniques under the information-theoretic approach, we see no reason to believe that these are the only, or best, families. Future research should explore the development of additional families of entropy-motivated branch question selection techniques. Perhaps the information-theoretic approach can be helpful in search guidance beyond branch question selection as well.

Appendix. Experimental results on MIPLIB problems

Our full results on the MIPLIB problems are given in Table 6. In each row, the strategy that performed the best is indicated by a bold-faced font. The numbers represent the number of nodes searched until an optimal solution was found and proven optimal, except where there is a percentage sign (%). This indicates that a 1 h time limit was reached before the search completed. The percentage given in the table is the *gap* (that is, the relative difference between the incumbent's value and the highest lower bound) at that time. Thus, a value of 0% would indicate that the optimal solution was found. None of the problems exhausted the main memory of the computer used in the experiments, so there should not be any issues with virtual memory performance.

It is particularly interesting to note instances where EB does better than (or at least as well as) SB since they perform roughly the same amount of computation per node. This occurs in 13 of the problem instances. This is quite remarkable, especially taking into account that EB does not use the objective function of the problem at all in making branching decisions.

The COMB(α, $1-\alpha$) heuristic did not work well for any of the values of α that we tried. The performance of this heuristic also had high variance. We did not include the data for this heuristic in the table of results.

References

[1] G. Nemhauser, L. Wolsey, Integer and Combinatorial Optimization, John Wiley & Sons, 1999.
[2] R. Karp, Reducibility among combinatorial problems, in: R.E. Miller, J.W. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, NY, 1972, pp. 85–103.
[3] P. Hart, N. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107.
[4] R. Dechter, J. Pearl, Generalized best-first search strategies and the optimality of A^*, Journal of the ACM 32 (3) (1985) 505–536.
[5] L. Wolsey, Integer Programming, John Wiley & Sons, 1998.
[6] M. Padberg, G. Rinaldi, Optimization of a 532-city symmetric traveling salesman problem by branch and cut, Operations Research Letters 6 (1987) 1–7.
[7] M. Padberg, G. Rinaldi, A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33 (1991) 60–100.
[8] ILOG Inc. CPLEX 9.1 User's Manual, 2005.
[9] T. Achterberg, T. Koch, A. Martin, Branching rules revisited, Operations Research Letters 33 (1) (2005) 42–54.
[10] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Finding cuts in the TSP (a preliminary report). Technical Report 95-05, DIMACS, Rutgers University, March 1995.
[11] V. Vazirani, Approximation Algorithms, Springer Verlag, 2001.
[12] C. Shannon, A mathematical theory of communication, The Bell System Technical Journal 27 (1948) 379–423. and 623–656.
[13] W. Glankwamdee, J. Linderoth, Lookahead branching for mixed integer programming, Technical Report 06T-004, Lehigh University, October 2006.
[14] R. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106.
[15] R. Bixby, S. Ceria, C. McZeal, M. Savelsbergh, An updated mixed integer programming library: MIPLIB 3.0, Optima 54 (1998) 12–15.
[16] A.J. Davenport, J. Kalagnanam, Price negotiations for procurement of direct inputs, Technical Report RC 22078, IBM, May 2001.
[17] T. Sandholm, S. Suri, Side constraints and non-price attributes in markets, Games and Economic Behavior 55 (2006) 321–330. Extended version in IJCAI-2001 Workshop on Distributed Constraint Reasoning.
[18] A. Andersson, M. Tenhunen, F. Ygge, Integer programming for combinatorial auction winner determination, in: Proceedings of the Fourth International Conference on Multi-Agent Systems, ICMAS, Boston, MA, 2000, pp. 39–46.
[19] T. Sandholm, S. Suri, A. Gilpin, D. Levine, CABOB: a fast optimal algorithm for winner determination in combinatorial auctions, Management Science 51 (3) (2005) 374–390.
[20] T. Sandholm, Optimal winner determination algorithms, in: P. Cramton, Y. Shoham, R. Steinberg (Eds.), Combinatorial Auctions, MIT Press, 2006, pp. 337–368. (Chapter 14).
[21] K. Leyton-Brown, Resource allocation in competitive multiagent systems, Ph.D. Thesis, Stanford University, August 2003.
[22] R. Gonen, D. Lehmann, Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics, in: Proceedings of the ACM Conference on Electronic Commerce, ACM-EC, Minneapolis, MN, October 2000, pp. 13–20.
[23] T. Sandholm, Expressive commerce and its application to sourcing: how we conducted $35 billion of generalized combinatorial auctions, AI Magazine 28 (3) (2007) 45–58.
[24] E.M.L. Beale, J.A. Tomlin, Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables, in: J. Lawrence (Ed.), Proceedings of the Fifth International Conference on Operational Research, Tavistock Publications, London, 1970, pp. 447–454.
[25] T. Sandholm, Algorithm for optimal winner determination in combinatorial auctions, Artificial Intelligence 135 (2002) 1–54.
[26] O. Bilde, J. Krarup, Sharp lower bounds and efficient algorithms for the simple plant location problem, Annals of Discrete Mathematics 1 (1977) 79–97.
[27] Y. Kochetov, D. Ivanenko, Computationally difficult instances for the uncapacitated facility location problem, in: Proceedings of the 5th Metaheuristic Conference, MIC 2003, Kyoto, Japan, 2003.
[28] J.H. Owen, S. Mehrotra, Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs, Computational Optimization and Applications 20 (2) (2001) 159–170.
[29] M. Karamanov, G. Cornuéjols, Branching on general disjunctions, Mathematical Programming A, Draft, July 2005, in press (doi:10.1007/s10107-009-0332-3).
[30] C. Gomes, M. Sellmann, Streamlined constraint reasoning, in: Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming, CP, Toronto, Canada, 2004, pp. 274–287.
[31] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, O. Vincent, Experiments in mixed-integer programming, Mathematical Programming 1 (1971) 76–94.
[32] A. Martin, Integer programs with block structure, Habilitation Thesis, Technical University of Berlin, Berlin, Germany, 1998.
[33] J.T. Linderoth, M.W.P. Savelsbergh, A computational study of search strategies for mixed integer programming, INFORMS Journal on Computing 11 (1999) 173–187.
[34] LINDO, API User's Manual, 2003.
[35] J. Patel, J.W. Chinneck, Active-constraint variable ordering for faster feasibility of mixed integer linear programs, Mathematical Programming 110 (2007) 1394–1411.
[36] J.R. Bitner, E.M. Reingold, Backtrack programming techniques, Communications of the ACM 18 (11) (1975) 651–656.
[37] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22 (4) (1979) 251–256.
[38] J. Etcheberry, The set-covering problem: a new implicit enumeration algorithm, Operations Research 25 (5) (1977) 760–772.
[39] D.M. Ryan, B.A. Foster, An integer programming approach to scheduling, in: A. Wren (Ed.), Computer Scheduling of Public Transport, North-Holland Publishing Company, 1981, pp. 269–280.
[40] D.M. Ryan, The solution of massive generalized set partitioning problems in aircrew rostering, Journal of the Operational Research Society 43 (5) (1992) 459–467.
[41] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, P.H. Vance, Branch-and-price: column generation for solving huge integer programs, Operations Research 46 (3) (1998) 316–329.
[42] T. Sandholm, S. Suri, BOB: improved winner determination in combinatorial auctions and generalizations, Artificial Intelligence 145 (2003) 33–58.