# Information-theoretic Approaches to Branching in Search*

**Andrew Gilpin**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

**Tuomas Sandholm**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

## Abstract

Deciding what question to branch on at each node is a key element of search algorithms. We introduce the information-theoretic paradigm for branching question selection. The idea is to drive the search to reduce uncertainty (entropy) in the current subproblem. We present four families of methods that fall within this paradigm. 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 other variables they govern. This significantly outperforms the state-of-the-art branching strategies. The fourth family is about branching using carefully constructed linear inequality constraints over *sets* of variables.

## 1 Introduction

Search is a fundamental technique for problem solving in AI and operations research (OR). 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.

We introduce a new paradigm for developing branching strategies, employing information theory as the 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 significantly improve speed over existing strategies. The rest of this section covers the needed background.

### 1.1 Integer programming

One of the most important computational problems in CS and OR is integer programming. Applications of integer programming include scheduling, routing, VLSI circuit design, and facility location [Nemhauser and Wolsey, 1999]. Integer programming is the problem of optimizing a linear function subject to linear constraints and integrality constraints on some of the variables. Formally:

**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 (the decision version of) MIP is $\mathcal{NP}$-complete [Karp, 1972], there are many sophisticated techniques that can solve very large instances in practice. We now review the existing techniques upon which we build our methods.

#### 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 (aka. 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 fast 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. 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* [Hart *et al.*, 1968], 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 [Dechter and Pearl, 1985]. (An almost identical node-selection strategy, *best-bound search*, is often used

*This was work was funded by, and conducted at, CombineNet, Inc., Fifteen 27th St., Pittsburgh, PA 15222.

in MIP [Wolsey, 1998].[1]) Therefore, in the experiments we will use best-bound search in all of the algorithms.

**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 [Padberg and Rinaldi, 1987; 1991], 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* [Nemhauser and Wolsey, 1999]. They are constraints that, when added to the problem at a search node, result in a tighter LP polytope (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 [ILOG Inc, 2005] is the 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, etc.) while allowing us flexibility in developing our own methods. (We used CPLEX version 9.1, *i.e.*, the newest version at the time of the experiments.) We configured the search order to best-bound, and we vary the branching strategies as we will discuss.

**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 used method in OR is to branch on the *most fractional variable*, *i.e.*, variable whose LP value is furthest from being integral [Wolsey, 1998]. Finding the branching variable under this rule is fast and this method yields small search trees on many problem instances.

A more sophisticated approach, which is better suited for certain hard problem instances, is *strong branching* [Applegate *et al.*, 1994]. 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.[2]

[1]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.

[2]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)

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* $z^l$ *be solution of current LP with* $x_i^l \leq \lfloor x_i \rfloor$.
   (c) *Let* $z^u$ *be solution of current LP with* $x_i^u \geq \lceil x_i \rceil$.
   (d) $score(x_i) \leftarrow 10 \cdot \min\{z^l, z^u\} + \max\{z^l, z^u\}$
3. $i^* \leftarrow \text{argmax}_{i \in candidates}\ score(x_i)$
4. *Return* $i^*$

There are many different ways that step 2.d could be performed in Algorithm 1. The above method is from [Applegate *et al.*, 1994]. We experimented with the following variations in addition to the method above:

1. $score(x_i) \leftarrow \min\{z^l, z^u\} + 10 \cdot \max\{z^l, z^u\}$
2. $score(x_i) \leftarrow z^l + z^u$
3. $score(x_i) \leftarrow \max\{z^l, z^u\}$
4. $score(x_i) \leftarrow \min\{z^l, z^u\}$
5. $score(x_i) \leftarrow (1 - x_i)z^l + x_i z^u$

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 [Applegate *et al.*, 1994].

## 2 The information-theoretic paradigm to branching in search

The simple key observation behind our paradigm is that in 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)[3]. Motivated by this observation, our paradigm is to guide the search so as to remove uncertainty from the nodes on the frontier of the search tree. In this paper we apply this paradigm 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 to this paradigm 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 behind our paradigm 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

[3]In addition to all variables getting fixed value assignments, a path can end by infeasibility (the assignments so far implying a contradiction) or by pruning by bound (the optimistic value of the node being no better than the value of the incumbent).

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. (Interpreting LP variables as probabilities is also used successfully in randomized approximation algorithms [Vazirani, 2001].)

Before we describe any specific families of branch question selection techniques under this paradigm, 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* [Shannon, 1948], 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:*

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

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 but found no major improvement compared to using $e(x)$ as in Definition 2. 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:*

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

*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 will have zero uncertainty according to this measure. We are now ready to present our four families of branching question selection techniques. They all fall under the information-theoretic paradigm 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* $\hat{x}^l_i \leq \lfloor x_i \rfloor$.
   (c) *Let* $\hat{x}^u$ *be solution vector of LP with* $\hat{x}^u_i \geq \lceil x_i \rceil$.
   (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. $i^* \leftarrow \text{argmin}_{i\in candidates}\ entropy(x_i)$
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 can be modified to perform more than one-step lookahead in the obvious way. This would likely lead to smaller search trees but more time would be spent per node. 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 [Quinlan, 1986]. 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 $x$% of

each other as tied. In the experiments (described below) we tested with $x \in \{0, 5, 10\}$.

### 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 $rank_{SB}(x_i)$ 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 $rank_{EB}(x_i)$ to be the rank of variable $x_i$ in terms of its EB entropy. Then, for each variable, we let $rank(x_i) = rank_{SB}(x_i) + rank_{EB}(x_i)$ and choose the variable $x_i$ with the smallest rank.

The second method, COMB($\rho$, $1-\rho$), computes a convex combination of the SB score (with weight $\rho$) 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.

**Experiments on MIPLIB 3.0**

MIPLIB [Bixby *et al.*, 1998] is a library of MIP instances that is commonly used for benchmarking MIP algorithms. We experimented on all instances of the newest MIPLIB (3.0).

EB and SB were comparable: they reached the 1-hour time limit on 34.7% and 24.5% of the instances, respectively. EB outperformed SB on 13 of the 49 instances. This is quite remarkable in the sense 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, showing that the combination is better than either of its parts.

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.

**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 implementation of SB was faster than CPLEX's default branching strategy. On those 121 instances, CPLEX's 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.[4] When limiting the size of the candidate list, we choose the candidates in order of most fractional.

| candidates | 10 iters | 25 iters | 100 iters | unlimited iters |
|---|---|---|---|---|
| 10 | 1947.00 | 2051.07 | 1966.21 | 1913.56 |
| unlimited | 1916.81 | 1962.09 | 1813.76 | 1696.53 |

Table 1: Average computation time over 121 instances for entropic branching with a 1-hour time limit.

| candidates | 10 iters | 25 iters | 100 iters | unlimited iters |
|---|---|---|---|---|
| 10 | 3600 | 3600 | 3600 | 3600 |
| unlimited | 2955.78 | 3195.58 | 1374.679 | 590.23 |

Table 2: Median computation time over 121 instances for entropic branching with a 1-hour time limit.

| candidates | 10 iters | 25 iters | 100 iters | unlimited iters |
|---|---|---|---|---|
| 10 | 62 | 65 | 62 | 61 |
| unlimited | 59 | 57 | 54 | 49 |

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

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 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

We introduce a third family of branching strategies, again within the entropy-based branching paradigm. This method

[4] 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.

is computationally less expensive than the methods we presented so far because it does not use lookahead. However, it does require an advanced knowledge of the structure of the problem. The problem with which we experiment is motivated by a real-world electronic commerce application: a combinatorial procurement auction (aka. reverse auction) where the buyer specifies the maximum number of winning suppliers [Davenport and Kalagnanam, 2001; Sandholm and Suri, 2006].

**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 [Sandholm and Suri, 2006]. Integer programming methods have been successfully used previously in winner determination research (*e.g.* [Andersson *et al.*, 2000; Sandholm *et al.*, 2005; Sandholm, 2006]), and we can very naturally formulate the above generalized winner determination problem as a MIP:

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

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 [Nemhauser and Wolsey, 1999]. 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*.) 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 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 supplier $j's$ bids. The branching strategy works as follows. For each supplier $j$ where $y_j \notin \{0, 1\}$, compute $entropy(j) = \sum_{i|s_i=j} e(x_i)$ and branch on the variable $y_{j'}$ where $j' = \text{argmin}_j \; entropy(j)$. 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)*.

### 5.2 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).[5] 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 [Sandholm, 2002] with $\alpha = 0.75$.[6] 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)).

| $s$ | $r$ | $m$ | $b$ | $k$ | **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% |

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.

## 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

[5] Furthermore, none of the combinatorial auction instance generators published in the literature have a notion of supplier.

[6] 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 $\alpha$ 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.

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 $x_i + x_j \leq 0$ and the other branch to be $x_i + x_j \geq 1$. In general, given a set $\mathcal{X}$ of variables and the current LP relaxation solution $\hat{x}$, we can let $k = \left\lfloor \sum_{i \in \mathcal{X}} \hat{x}_i \right\rfloor$ and we can generate the branches $\sum_{i \in \mathcal{X}} x_i \leq k$ and $\sum_{i \in \mathcal{X}} x_i \geq k + 1$.[7,8,9] Then, 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.[10] 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.)

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 (*e.g.*, from conflicts in subproblems [Achterberg, 2006; Sandholm and Shields, 2006]), 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 two, then this equality applies to the number of nodes in the tree as well.*

PROOF. There are $\prod_{i=1}^{n} |D_i|$ solutions. For any nontrivial branching, that is the number of leaves. Binary 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). We limited our algorithm to considering branches containing just 1 or 2 variables in order to keep the number of candidate branching questions 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 we found that 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 Conclusions, discussion, and future research

We introduced a new paradigm for branch selection in search based on an information-theoretic approach. In the beginning of a search, there is the most uncertainty about the optimal solution. When the search is complete, there is zero uncertainty. Using this observation, we developed four families of methods for selecting what question to branch on at a search node so as to reduce the amount of uncertainty in the search process. All four of our families of methods 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. The fourth family is about branching using carefully constructed linear inequality constraints over *sets* of variables.

One can view many existing search methods as quick approximations of entropy-based search. First, the classic OR idea of branching on the variable that has the most fractional LP value at the node in best-bound search [Wolsey, 1998] is

[7] 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.

[8] The special case $k = 0$ of this branching question has been shown to be effective in set partitioning, set packing, and set covering problems [Etcheberry, 1977; Ryan and Foster, 1981; Ryan, 1992]. For such 0-1 problems, theory has been developed that shows that each path consisting of carefully selected such branches will yield an LP with all vertices integral after a quadratic number of branches in the number of constraints [Barnhart *et al.*, 1998]. Future research includes exploring the use of techniques from our Family 4 to select among such branching candidates in those problems.

[9] More generally, one can branch on the disjunction $\sum_{i \in \mathcal{X}} a_i x_i \leq r$ versus $\sum_{i \in \mathcal{X}} a_i x_i \geq r + 1$, where the constants $a_i$ can be positive or negative, as long as there are no integer solutions between those two hyperplanes (*e.g.*, [Owen and Mehrotra, 2001]). 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 [Gomes and Sellmann, 2004], and it would be interesting to try the information-theoretic paradigm for branch question selection in that context as well.

[10] 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 nontermination with any branching rule that is based on location information. (In fact, we do not even conduct the lookahead for such $\mathcal{X}$.)

a lookahead-free approximation to 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* [Bitner and Reingold, 1975], and its usual tie-breaker, the *most-constraining-variable-first* heuristic [Brélaz, 1979], 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.

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).

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

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 which the algorithm knows the right answer with high confidence [Sandholm *et al.*, 2005; Sandholm, 2006]. 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. One could also try our paradigm 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 paradigm, 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 paradigm can be helpful in search guidance beyond branch question selection as well.

## References


[Achterberg, 2006] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 2006. To appear.

[Andersson *et al.*, 2000] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS*, pp. 39–46, Boston, MA, 2000.

[Applegate *et al.*, 1994] David Applegate, Robert Bixby, Vasek Chvátal, and William Cook. Finding cuts in the TSP. Technical Report 95-05, DIMACS, Rutgers University, March 1995.

[Barnhart *et al.*, 1998] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3):316–329, 1998.

[Bitner and Reingold, 1975] James Bitner and Edward Reingold. Backtrack programming techniques. *CACM*, 18(11):651–656.

[Bixby *et al.*, 1998] Robert Bixby, Sebastian Ceria, Cassandra McZeal, and Martin Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 54:12–15, 1998.

[Brélaz, 1979] Daniel Brélaz. New methods to color the vertices of a graph. *CACM*, 22(4):251–256, April 1979.

[Davenport and Kalagnanam, 2001] Andrew J. Davenport and Jayant Kalagnanam. Price negotiations for procurement of direct inputs. Technical Report RC 22078, IBM, May 2001.

[Dechter and Pearl, 1985] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[Etcheberry, 1977] Javier Etcheberry. The set-covering problem: A new implicit enumeration algorithm. *Operations Research*, 25(5):760–772, September–October 1977.

[Gomes and Sellmann, 2004] Carla Gomes and Meinolf Sellmann. Streamlined constraint reasoning. In *CP*, pp. 274–287, 2004.

[Hart *et al.*, 1968] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Sys. Sci. and Cybernetics*, 4(2):100–107.

[ILOG Inc, 2005] ILOG Inc. CPLEX 9.1 User's Manual, 2005.

[Karp, 1972] Richard Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pp. 85–103. Plenum Press, NY.

[Nemhauser and Wolsey, 1999] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1999.

[Owen and Mehrotra, 2001] Jonathan H. Owen and Sanjay Mehrotra. Experimental results on using general disjunctions in branch-and-bound for general-integer linear programs. *Computational Optimization and Applications*, 20(2):159–170, November 2001.

[Padberg and Rinaldi, 1987] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Oper. Res. Letters*, 6:1–7, 1987.

[Padberg and Rinaldi, 1991] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[Quinlan, 1986] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Ryan and Foster, 1981] David Ryan and Brian Foster. An integer programming approach to scheduling. In A. Wren, ed., *Computer Scheduling of Public Transport*, pp. 269–280. North-Holland.

[Ryan, 1992] David Ryan. The solution of massive generalized set partitioning problems in aircrew rostering. *The Journal of the Operational Research Society*, 43(5):459–467, May 1992.

[Sandholm and Shields, 2006] Tuomas Sandholm and Robert Shields. Nogood learning for mixed integer programming. Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal, Sept. 18–22, 2006.

[Sandholm and Suri, 2006] Tuomas Sandholm and Subhash Suri. Side constraints and non-price attributes in markets. *Games and Economic Behavior*, 55:321–330, 2006.

[Sandholm *et al.*, 2005] Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for winner determination in combinatorial auctions. *Management Science*, 51(3):374–390, 2005.

[Sandholm, 2002] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, January 2002.

[Sandholm, 2006] Tuomas Sandholm. Optimal winner determination algorithms. In P. Cramton, Y. Shoham, and R. Steinberg, eds., *Combinatorial Auctions*, pp. 337–368. MIT Press, 2006.

[Shannon, 1948] Claude Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423 and 623–656.

[Vazirani, 2001] V. Vazirani. *Approximation Algorithms*. Springer.

[Wolsey, 1998] Laurence Wolsey. *Integer Programming*. Wiley.