

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 to branch on at each node is a key element of search algorithms. We present four families of methods for selecting what question to branch on. They are all information-theoretically motivated to *reduce uncertainty in remaining sub-problems*. In the first family, a good variable to branch on is selected based on lookahead. In real-world procurement optimization, this entropic branching method outperforms default CPLEX and strong branching. The second family combines this idea with strong branching. The third family does not use lookahead, but instead exploits features of the underlying structure of the problem. Experiments show that this family significantly outperforms the state-of-the-art branching strategy when the problem includes indicator variables as the key driver of complexity. 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. In this paper we introduce a new paradigm for developing branching strategies, employing information theory as the principle that guides the development. In the context of solving integer programs, we develop four high-level families of strategies, and show that some of those families significantly improve performance over existing strategies. All four strategies aim to reduce the uncertainty in the current subtree.

The rest of this section introduces the necessary background material. Each of the following four sections introduces one high-level family of new strategies.

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] (aka. *best-bound* search [Wolsey, 1998]), the leaf with the lowest lower bound is expanded next. A* is desirable in the sense that no optimal tree search algorithm can guarantee expanding fewer nodes [Dechter and Pearl, 1985]. Therefore, in all of the experiments in this paper, we will use A* as the search strategy for 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, 2002] is a commercial software product for solving MIPs. It can be configured to support many different branching algorithms, including branch-and-cut. In addition, CPLEX makes available low-level interfaces for controlling the search. We configured the search order to A* and we vary the branching strategies as we will discuss. 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.

Strong branching

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). One commonly used method is to branch on the variable whose value is furthest from being integral [Wolsey, 1998]. In other words, the question is: “What should the value of this variable be?”, and the children correspond to different answers to this question. For many problem instances, this approach is effective.

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.¹

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. $i^* \leftarrow \operatorname{argmax}_{i \in candidates} score(x_i)$
4. Return i^*

There are many different ways that step 2.d could be performed in Algorithm 1. The above method is from [Applegate

¹Often 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.

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 Entropic lookahead for variable selection

The main idea behind our approaches is to treat the fractional portion of integer-constrained variables as probabilities, indicating the probability with which we expect the variable to be greater than its current value in the optimal solution. Using this reasoning, our branching heuristic is designed to guide the search so as to remove uncertainty from the partial solution in the current search node. For any search algorithm, the beginning of the search is where there is the most uncertainty about the optimal solution to the problem. When the search is complete, there is zero uncertainty as the optimal solution has been found. Our branching heuristic is designed to direct the search from the root node (with high uncertainty) to the optimal node (with zero uncertainty) in as direct a way as possible. 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* in potential child nodes, choosing a variable to branch on that yields children with the least uncertainty.

While there has been much work on developing branching heuristics, to our knowledge this is the first work that takes an information-theoretic approach to guiding the search process.

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.²

Before we describe our approach to branching in search, 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.

²Interpreting LP variables as probabilities is also used successfully in randomized approximation algorithms [Vazirani, 2001].

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. 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 be using this computation for $e(x)$.

Entropy is additive for independent variables so we can 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 generally possible for there to be multiple optimal solutions in an optimization problem, it is worth noting that all optimal 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: $x - \lfloor x \rfloor$.

Now we are ready to present our variable selection method:

Algorithm 2 Entropic Branching (EB)

1. $\text{candidates} \leftarrow \{i \mid x_i \text{ fractional}\}$
2. For each $i \in \text{candidates}$:
 - (a) $x_f \leftarrow x_i - \lfloor x_i \rfloor$
 - (b) Let \hat{x}^l be solution vector of LP with $\hat{x}_i^l \leq \lfloor x_i \rfloor$.
 - (c) Let \hat{x}^u be solution vector of LP with $\hat{x}_i^u \geq \lceil x_i \rceil$.
 - (d) $\text{entropy}(x_i) \leftarrow \sum_{j=1}^n (1 - x_f) e(\hat{x}_j^l) + x_f e(\hat{x}_j^u)$
3. $i^* \leftarrow \text{argmin}_{i \in \text{candidates}} \text{entropy}(x_i)$
4. Return i^*

EB is a *general* method, usable on any MIP as it does not make any assumptions about the underlying model for the problem on which it is used. EB (and SB) could be modified to perform more than one-step lookahead in a straightforward way. However, we suspect that this increased computational effort would not reduce tree size enough to be beneficial.

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.

3 Combining 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 combinatorial.

3.1 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 can also consider two variables having SB scores within $x\%$ of each other as tied. For a given percentage x , we refer to this method as TB($x\%$).

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

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

3.3 Experiments on these lookahead-based branching strategies

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 is never an issue.

Experiments on MIPLIB

MIPLIB [Bixby *et al.*, 1998] is a library of MIP instances that is commonly used for benchmarking integer programming algorithms. We performed extensive experiments on the MIPLIB problem instances.

candidates	10 iters	25 iters	100 iters	∞ iters
10	1947.00	2051.07	1966.21	1913.56
∞	1916.81	1962.09	1813.76	1696.53

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

candidates	10 iters	25 iters	100 iters	∞ iters
10	3600	3600	3600	3600
∞	2955.78	3195.58	1374.679	590.23

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

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 the simpler strategy of branching on the most fractional variable (although that strategy 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 SB performs well compared to simple branching strategies, and we demonstrate the further improvement obtainable by using EB 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 problems set on which SB is the algorithm of choice (compared to non-lookahead-based standard techniques). We acquired 121 large-scale real-world industrial procurement optimization instances, of varying sizes and structures. On those instances, CPLEX’s implementation of SB is on average 27% faster than the default search algorithm of CPLEX. Thus we concluded that SB is a good algorithm for this data set, and performed a comparison of EB against SB on this data.

In order to be able to carefully and fairly control the parameter settings of both SB and EB, we implemented both. (Using CPLEX’s 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.

Tables 1–3 summarize the experimental results for EB. We varied the size of the candidate list and the number of simplex iterations performed at each search node. 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

candidates	10 iters	25 iters	100 iters	∞ iters
10	62	65	62	61
∞	59	57	54	49

Table 3: Total number of instances (out of 121) requiring more than one hour.

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 performance improvements.

EB was 29.5% faster than SB: the comparable number to EB’s 1696.53 average seconds was 2406.07 for SB.

4 Entropic lookahead-free variable selection

In this section we introduce a third family of branching strategies, again within the entropy-based branching paradigm. This method is computationally less expensive than the methods we have presented thus 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 $\mathcal{M} = \{1, \dots, m\}$ be the m goods that the buyer wishes to procure (the buyer wants at least one unit of each good). Let $\mathcal{S} = \{1, \dots, s\}$ be the s suppliers participating in the procurement and let $\mathcal{B} = \{B_1, \dots, B_n\}$ 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.

Integer programming methods have been successfully used previously in winner determination research [Andersson *et al.*, 2000]. We can very naturally formulate the above generalized winner determination problem as a MIP:

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

This problem is \mathcal{NP} -complete, even if the bids are on single items only [Sandholm and Suri, 2006]. 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] (for example an implication that if an indicator is zero, then each of a set of other variables has to be zero). Constraints that state that at most (or exactly) k variables from a set of variables can be nonzero are an important special case. Typically, the LP relaxation gives poor approximate values for the indicator variables, and as we show, the branching method that we propose helps significantly to address this problem.

4.1 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 $j' = \operatorname{argmin}_j entropy(j)$. 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)*.

4.2 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).³ 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 with $\alpha = 0.75$.⁴

For this data distribution, we determined that CPLEX's default strategy of branching on most-fractional variables first was the strongest strategy (in particular, it was faster than strong branching on this data). Table 4 shows experimental results comparing IEB with CPLEX using that branching

³Furthermore, none of the combinatorial auction instance generators published in the literature have a notion of supplier.

⁴Given a parameter α , $0 < \alpha < 1$, we generate each bid in the Decay distribution [Sandholm, 2002] 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.

strategy. The results indicate that IEB performs significantly better than the state-of-the-art branching strategy.

s	r	m	b	k	CPLEX	IEB
20	10	10	100	5	25.63	11.81
30	15	15	150	8	5755.92	551.82
40	20	20	200	10	37.05%	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.

5 Entropic lookahead for multi-variable branches

In this section we introduce a fourth family of methods for determining good questions 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. It is known that 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 $k = \lfloor \sum_{i \in \mathcal{X}} \hat{x}_i \rfloor$ and we can generate the branches $\sum_{i \in \mathcal{X}} x_i \leq k$ and $\sum_{i \in \mathcal{X}} x_i \geq k + 1$. No other value of k is worth considering; any other integer value would cause one child to be exactly the same as the node. Then, instead of branching on the variable with the smallest amount of entropy in its child nodes, we select the *set* of variables for branching that results in the smallest amount of entropy in the two child nodes.⁵ 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. (For example, 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 *Ignoring the effects of pruning, the search tree size (both in the number of nodes and number of leaves) 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).*

⁵Branching on a set of variables \mathcal{X} is not helpful 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 at all, so the corresponding child node will be identical to its parent. Thus, if the sum is integral, we do not consider \mathcal{X} as a branching candidate (we do not even conduct the lookahead for it).

PROOF. There are 2^n solutions, and each solution corresponds to a leaf. For any branching, there are 2^n leaves. Binary trees with the same number of leaves have the same number of nodes. \square

We performed experiments on the MIPLIB data and on combinatorial auction winner determination problem instances (from the Decay distribution [Sandholm, 2002]). 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). 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.

6 Discussion

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 significantly better than strong branching and default CPLEX on real-world procurement optimization. The second family combines this idea with strong branching in different ways. The third family does not use lookahead. Experiments show that this family significantly outperforms the state-of-the-art branching strategy when the problem includes indicator variables as the key driver of complexity. The fourth family is about branching using carefully constructed linear inequality constraints over sets of variables.

One can think of many existing search methods as quick approximations to entropy-based search. Using entropy as the cost-to-go function (*i.e.*, *h*-function) in A* search yields depth-first-search. The classic OR idea of branching on the variable that has the most fractional LP value at the node in A* search [Wolsey, 1998] can be viewed as 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. Finally, in constraint satisfaction problems, the usual most-constrained-variable-first heuristic [Russell and Norvig, 2003] approximates entropy-based variable selection: assigning a highly constrained variable affects the other variables the most, thus reducing the entropy of those variables.

Future research includes testing these ideas on additional problems, generating quick techniques for curtailing the set of candidate questions to branch on before lookahead, and investigating new entropy-motivated search methods.

References

- [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. The traveling salesman problem. Technical report, Rutgers University, DIMACS, 1994.
- [Bixby *et al.*, 1998] Robert Bixby, Sebastian Ceria, Cassandra McZel, and Martin Savelsbergh. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 54:12–15, 1998.
- [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.
- [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, 2002] ILOG Inc. CPLEX 8.1 Reference Manual, 2002.
- [Karp, 1972] Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.
- [Nemhauser and Wolsey, 1999] George Nemhauser and Laurence Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.
- [Padberg and Rinaldi, 1987] Manfred Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations 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 TSPs. *SIAM Review*, 33:60–100, 1991.
- [Quinlan, 1986] Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [Russell and Norvig, 2003] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
- [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, 2002] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, January 2002. Early version: IJCAI-99.
- [Shannon, 1948] Claude Shannon. A mathematical theory of communication. *Bell System Tech. J.*, 27:379–423, 623–656, 1948.
- [Vazirani, 2001] Vijay Vazirani. *Approximation Algorithms*. Springer Verlag, 2001.
- [Wolsey, 1998] Laurence Wolsey. *Integer Programming*. John Wiley & Sons, 1998.