The Accuracy of Search Heuristics:
An Empirical Study on Knapsack Problems*

Daniel H. Leventhal and Meinolf Sellmann

Brown University
Department of Computer Science
115 Waterman Street, P.O. Box 1910
Providence, RI 02912
{dlev,sello}@cs.brown.edu

Abstract. Theoretical models for the evaluation of quickly improving search strategies, like limited discrepancy search, are based on specific assumptions regarding the probability that a value selection heuristic makes a correct prediction. We provide an extensive empirical evaluation of value selection heuristics for knapsack problems. We investigate how the accuracy of search heuristics varies as a function of depth in the search-tree, and how the accuracies of heuristic predictions are affected by the relative strength of inference methods like pruning and constraint propagation.

1 Motivation

The modern understanding [13] of systematic solvers for combinatorial satisfaction and optimization problems distinguishes between two fundamentally different principles. The first regards the intelligent reasoning about a given problem or one of its subproblems. In an effort to reduce the combinatorial complexity, solvers try to assess whether a part of the solution space can contain a feasible or improving solution at all. Moreover, when there is no conclusive evidence with respect to the global problem, solvers try to eliminate specific variable assignments that can be shown quickly will not lead to improving or feasible solutions. The principle of reasoning about a problem is called *inference* and is comprised of techniques like relaxation and pruning, variable fixing, bound strengthening, and, of course, constraint filtering and propagation. To strengthen inference, state-of-the-art solvers also incorporate methods like the computation of valid inequalities and, more generally, no-good learning and redundant constraint generation, as well as automatic model reformulation.

While inference can be strengthened to a point where it is capable of solving combinatorial problems all by itself (consider for instance Gomory's cutting plane algorithm for general integer problems [11] or the concept of k-consistency in binary constraint programming [8,3]), today's most competitive systematic solvers complement the reasoning about (sub-)problems with an active *search* for solutions. The search principle is in some sense the opposite of inference (see [13]): When reasoning about a problem we consider the solution space as a whole or even a relaxation thereof (for example by considering only one constraint at a time or by dropping integrality constraints). In

---

search, on the other hand, we actually zoom into different parts of the space of potential solutions. Systematic solvers derive their name from the fashion in which this search is conducted. Namely, the space of potential solutions is partitioned and the different parts are searched systematically. This is in contrast to non-systematic solvers that are based on, for example, local search techniques. The main aspects of systematic solvers are how the solution space is partitioned, and in what order the different parts are to be considered. Both choices have an enormous impact on solution efficiency.

From an intellectual standpoint, we have every right to distinguish between search and inference in the way we described before. They are two different principles and constitute orthogonal concepts that each can be applied alone or in combination with the other. However, it has long been observed that (when applied in combination as practically all successful systematic solvers in constraint programming, satisfiability, and mathematical programming do) inference and search need to be harmonized to work together well (see, e.g. [2]).

Information provided by inference techniques can also be used to organize and guide the search process. For example, in integer linear programming fractional solution values are often used to determine whether a variable should be rounded up or down first. Such heuristics are commonly compared with each other based on some specific efficiency measure that is global to the specific approach, such as total time for finding and proving an optimal solution, or the best solution quality achieved after some hard time-limit.

However, to our knowledge very little research has been conducted which investigates how *accurate* information taken from inference algorithms actually is, or how this accuracy evolves during search. This is quite surprising given that many theoretical studies need to make certain assumptions about the relative correctness of search heuristics or the dependency of heuristic accuracy and, for instance, the depth in the search tree. The theoretical model that is considered in [20] to explain and study heavy-tailed runtime distributions, for example, is based on the assumption that the heuristic determining how the solution space is to be partitioned has a fixed probability of choosing a good branching variable. Moreover, Harvey and Ginsberg's limited discrepancy search (LDS) [12] is based on the assumption that the probability that the value selection heuristic returns a good value is constant (whereby, at the same time, they also assume that heuristics are marginally more likely to fail higher up in the tree than further below). Walsh's depth-bounded discrepancy search (DDS) [19], on the other hand, is implicitly based on the assumption that value selection heuristics are relatively uninformed at the top of the tree while the error probability (as a function of depth) converges very quickly to zero. Moreover, theoretical models for both LDS and DDS assume that there is a constant probability throughout search that the heuristic choice matters at all (which it does not when all subtrees contain (best) solutions that are equally good).

1.1   A Disturbing Case Study

Commonly, heuristics are evaluated on their overall performance in the context of a specific algorithm. In the following example, we show what can happen when heuristic accuracy is inferred from global efficiency.

Assume that we need to quickly compute high-quality solutions to knapsack problems. We have several heuristics and compare their performance when employed within
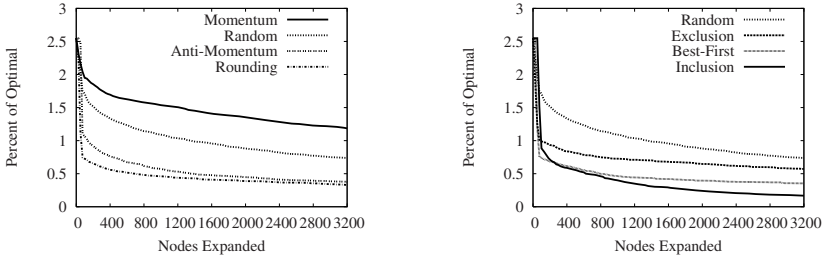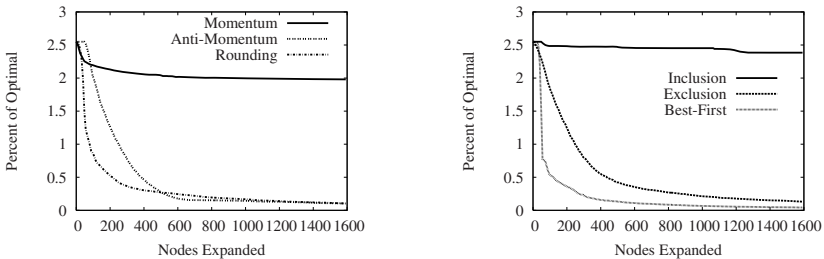
Fig. 1. Average optimality gap over number of search nodes when running PA with different heuristics on 500 almost strongly correlated instances with 50 items each. The curves are split into two graphs for better readability.

a depth-first search (DFS). Figure 1 shows how the average optimality gap, that is, the gap between current best solution and the optimal solution value, evolves when executing the algorithm with different heuristics on almost strongly correlated knapsack instances. (We will explain the heuristics and what strongly correlated knapsack instances are later - for now their exact definition is not important.) We see a clear separation: Heuristics inclusion, rounding, and best-first make very rapid improvements, while exclusion and momentum offer no significant gains over a random choice, or perform even worse.

In order to find a high-quality solution quickly, we clearly cannot afford a complete search. Therefore, we need to decide which parts of the search space we want to consider. LDS was developed exactly for this purpose. The strategy tries to guide the search to those parts of the search space that are most likely to contain feasible (in case of constraint satisfaction) or very good (in case of constraint optimization) solutions. Based on our findings in Figure 1, our recommendation would be to combine LDS with the inclusion heuristic.

The depressing result of this recommendation is shown in Figure 2: The choice of using the inclusion heuristic, whose predictions we so trusted, is the worst we could have made!

Fig. 2. Average optimality gap over number of search nodes when running PA with limited discrepancy search and different heuristics on 500 almost strongly correlated instances with 50 items each

1.2   A Case for the Direct Assessment of Heuristic Accuracy

Clearly, we lack some fundamental understanding of heuristic decision making. Obviously, it is not enough to design a heuristic that makes intuitive sense and to evaluate it within one specific algorithm. In this paper, we argue that *direct measurement* is needed on how accurate heuristic predictions are, how their accuracy differs with respect to the state of the search, and how it is affected by inference. One of the few examples of such research was presented in [10]: On the maximum satisfiability problem, Gomes et al. compare the accuracy of value selection heuristics based on a linear relaxation on one hand or a semi-definite relaxation on the other in correlation to the level of constrainedness of the given instances. Another example is given in [16] where the accuracy of relaxation-based search heuristics is found to depend on the sparsity of the linear program. We firmly believe that the first step towards the development of superior search algorithms must be a thorough empirical study how the accuracy of search heuristics depends on the current state of the search and how they interact with the inference mechanisms that are being used. With this paper, we wish to make a first step in that direction.

In the following section, we review the knapsack problem and describe a complete algorithm to tackle it. Before we present the results of our extensive experimentation that amounts to roughly 12,000 CPU hours on a dual-core 64-bit 2.8GHz Intel-Xeon processor, in Section 3 we introduce two performance measures for value selection heuristics. In Section 4, we investigate how the accuracies of different value selection heuristics behave as a function of depth in the search tree. Finally, we study how inference mechanisms like cutting plane generation and constraint propagation can influence the accuracy of those heuristics in Section 5.

2   Knapsack Problems

We begin our presentation by introducing the knapsack problem that will serve as the basis of our experimentation.

Definition 1 (Knapsack Problem). *Given a natural number $n \in \mathbb{N}$, we denote by $p_1, \ldots, p_n$ the* profits*, and by $w_1, \ldots, w_n$ the* weights *of n items. Furthermore, given a* knapsack capacity $C \in \mathbb{N}$, *the knapsack problem* consists in finding a subset of items $S \subseteq \{1, \ldots, n\}$ *such that the total weight of the selection does not exceed the knapsack capacity, i.e. $\sum_{i \in S} w_i \leq C$, and the total profit of the selection $\sum_{i \in S} p_i$ is maximized.*

We chose the knapsack problem as our target of investigation as it has some very desirable properties.

- First, the problem has a simple structure and is yet NP-hard [9].
- The knapsack problem is of great practical relevance as it can be viewed as a relaxation of very many real-world problems. Whenever we maximize some additive profit while a limited resource is exhausted linearly, knapsacks are somewhere hidden in the problem.
- Moreover, there exist many value selection heuristics for the problem so that we are in a position to base our study on more than just one special heuristic.

– Finally, for our experiments we need a large set of problem instances so that we can gather meaningful statistics. There exist knapsack generators that create instances in different classes that are well-documented and well-studied in the literature.

With respect to our last reason for considering knapsacks, we will exploit the following four benchmark classes of knapsack instances that are widely used in the knapsack literature [15]. In all cases, profits and weights are limited to the interval $[1, 10^6]$. When the procedure below results in an assignment of a value outside of this interval, we set the corresponding weight or profit to the closest interval bound.

– In **uncorrelated** instances, profits and weights of all items are drawn uniformly in $[1, 10^6]$ and independently from one another.
– **Weakly correlated** instances are generated by choosing profits uniformly in a limited interval around the corresponding weights. More precisely, after choosing a random weight for each item, we select the profit p_i uniformly from the interval $[w_i - 10^5, w_i + 10^5]$.
– Instances are **strongly correlated** when all profits equal their item's weight plus some constant. In our algorithm, we set $p_i = w_i + 10^5$.
– In **almost strongly correlated** instances, profits are chosen uniformly from a very small interval around the strongly correlated profit. Precisely, we choose p_i uniformly in $[w_i + 10^5 - 10^3, w_i + 10^5 + 10^3]$.

2.1 Branch and Bound for Knapsacks

Knapsacks with moderately large weights or profits are most successfully solved by dynamic programming approaches [14]. Once both profits and weights become too large, however, excessive memory requirements make it impossible to employ a dynamic program efficiently. Then, it becomes necessary to solve the problem by standard branch-and-bound techniques. As is common in integer linear programming, the bound that is used most often is the linear continuous relaxation of the problem. According to Dantzig, for knapsacks it can be computed very quickly [5]: First, we arrange the items in non-increasing order of efficiency, i.e., $p_1/w_1 \geq \cdots \geq p_n/w_n$. Then, we greedily select the most efficient item, until doing so would exceed the capacity of the knapsack, i.e., until we have reached the item s such that $\sum_{i=1}^{s-1} w_i \leq C$ and $\sum_{i=1}^{s} w_i > C$. We say that s is the *critical item* for our knapsack instance. We then select the maximum fraction of item s that can fit into the knapsack, i.e., $C - \sum_{j=1}^{s-1} w_j$ weight units of item s. The total profit of this relaxed solution is then $\tilde{P} := \sum_{j=1}^{s-1} p_j + \frac{p_s}{w_s}(C - \sum_{j=1}^{s-1} w_j)$.

Pruning the Search
Based on the relaxation, we can terminate the search early whenever we find that one of the following conditions holds [17]: *Optimality:* If no fractional part of the critical item can be added, then the optimal relaxed solution is already integral and therefore solves the given (sub-)problem optimally. *Insolubility:* If the capacity C of the (remaining) knapsack instance is negative, then the relaxation has no solution. *Dominance:* If the profit of the relaxed solution is not bigger than the value of any lower bound on the best integer solution, then the (sub-)tree under investigation cannot contain any improving solution. In all three cases, we may backtrack right away and thereby *prune* the search tree of branches that do not need explicit investigation.

For the sake of dominance detection, we need a lower bound, and the better (i.e. larger) that bound, the stronger our ability to prune the search. Obviously, every time we find an improving knapsack solution, we will update that lower bound. To speed up the search, however, we can do a little more: At the root-node, we compute an initial lower bound by taking all the profit of all non-fractional items of the initial relaxed solution. To strengthen this bound, we fill the remaining capacity with the remaining items (the ones after the critical item in the efficiency ordering), skipping subsequent items that fit only fractionally, until we either run out of capacity or items.

Organization of Search

We have outlined our approach in regard to inference. Now, with respect to search: At every search node, we choose to either include or exclude the critical item. This choice of the branching variable ensures that the upper bound that we compute has a chance of tightening in both children that are generated simultaneously. Note that this would not be the case if we would branch over any other item. Then, for at least one of the two child-nodes the upper bound would stay exactly the same as before. The value selection heuristics that we will study empirically are the following:

- **Random Selection Heuristic:** Choose at random the subtree that is investigated first. The reason why we may believe in this strategy may be that we feel that there are no good indicators for preferring one child over the other.
- **Inclusion Heuristic:** Always choose to insert the critical item first. A justification to consider this heuristic is that solutions with high total profits must include items.
- **Exclusion Heuristic:** Always choose to exclude the critical item first. This heuristics makes sense as it assures that no item with greater efficiency needs to be excluded to make room for the critical item.
- **Rounding Heuristic:** Consider the fractional value of the critical item: If it is at least 0.5, then include the critical item, otherwise try excluding it first. This heuristic makes intuitive sense when we trust the linear continuous relaxation as our guide.
- **Momentum Heuristic:** See whether the fractional value of the critical item is larger or lower than what its value was at the root-node. If the value has decreased, then exclude the item first, otherwise include it first. This heuristic is motivated by observing that the history of fractional values builds up a kind of momentum with respect to the "correct" value of the branching variable.
- **Anti-Momentum Heuristic:** The exact opposite of the momentum heuristic.
- **Best-First Heuristic:** Compute the relaxation value of both children and then consider the one with the higher value first. The intuition behind it is that, in a complete method, we will need to consider the better child anyway as there is no chance for the worse child to provide a lower bound that allows us to prune the better child.

2.2   Stronger Inference: Knapsack Cuts and Knapsack Constraints

The previous paragraphs sketch the baseline branch-and-bound algorithm that we will employ to study the different value selection heuristics. One question that we are interested in answering with this paper is how inference influences search heuristics. To this end, we consider two ways of strengthening inference for knapsacks.

The first is to compute a stronger upper bound. By adding valid inequalities known as "knapsack cuts," we can strengthen the linear continuous relaxation and thereby

obtain a tighter bound for pruning [4]. The redundant constraints are based on the following observation: When computing the upper bound, we find a subset of items $S := \{1, \ldots, s\}$ out of which it is infeasible to include more than $s - 1$ in the knapsack, because this would exceed the limited capacity. As a matter of fact, out of the set of items $S' := S \cup \{i > s \mid w_i \geq w_j, \; \forall \, j \in S\}$, we can select at most $s - 1$ items. Therefore, it is legitimate to add the inequality $\sum_{i \in S'} X_i \leq s - 1$, where X_i is a binary variable which is 1 iff item i is included in the knapsack. We can add more such redundant constraints by adding one of them for each item skipped during the lower bound computation, because for each of these items we find a new set of items that, when added as a whole, would overload the knapsack.

The second way to strengthen inference for knapsacks is to add filtering to our baseline approach. That is, on top of the pruning based on upper and lower bound, we can also try to determine that certain items must be included in (or excluded from) any improving solution. For a given lower bound B, the corresponding constraint is true if and only if $\sum_i w_i X_i \leq C$ while simultaneously $\sum_i p_i X_i > B$. Achieving generalized arc-consistency for this global constraint is naturally NP-hard. However, we can achieve relaxed consistency with respect to Dantzig's bound in amortized linear time [6].

Now, adding solely the knapsack constraint that directly corresponds to the given knapsack instance is only of limited interest as constraint filtering draws its real strength from constraint propagation, i.e. the exchange of information between constraints that reflect different views of the problem. By exploiting the knapsack inequalities that we just discussed, we can add a couple of other knapsack constraints. At the root node, we compute the linear relaxation of the knapsack instance augmented by knapsack cuts. For the constraint $\sum_i w_i X_i \leq C$ we obtain a dual multiplier π_0. The same holds for the redundant constraints: For each cutting plane $\sum_{i \in I_r} X_i \leq s_r$, $1 \leq r \leq R$, we obtain a dual multiplier π_r. As suggested in [7], we use those multipliers to coalesce the different constraints into one new knapsack constraint

$$\sum_i p_i > B \qquad \text{and} \qquad \sum_i (\pi_0 w_i + \sum_{r \mid i \in I_r} \pi_r) X_i \leq \pi_0 C + \sum_r \pi_r s_r.$$

Finally, as was suggested in [18], we generate more redundant constraints by choosing a multiplier $\lambda \in \mathbb{N}$ and combining the different constraints: We multiply each constraint with a new multiplier that is a factor λ larger than the previous one (i.e., the first constraint is multiplied with 1, the next multiplier is λ, the next λ^2, and so on) and then we sum them all up. As this would quickly lead to knapsacks with extremely large item weights, we restrict ourselves to random selections of at most m out of the $R + 1$ inequalities, whereby an inequality is added to the selection with a probability that is proportional to the optimal dual multiplier at the root-node. For our experiments, we chose $\lambda = 5$ and $m = 5$, and we generate ten constraints in this way.

So with the original knapsack constraint and the knapsack based on the optimal dual multipliers, we have a total of twelve constraints that we filter and propagate at every choice point until none of the constraints can exclude or include items anymore.

In summary, to evaluate how different inference techniques can influence the performance of search heuristics, we consider four different approaches in our experiments:

– **Pure Approach (PA):** Our baseline approach conducts branch and bound based on the simple linear continuous upper bound of the knapsack problem computed by

Dantzig's efficiency sorting procedure. The critical item determines the branching variable, the value selection heuristic is given as a parameter.

– **Valid Inequality Approach (VIA):** A variant of PA where we use the linear continuous relaxation augmented by knapsack cuts to determine upper bounds which is computed by the simplex algorithm. The branching variable is determined as the fractional variable that corresponds to the item with greatest efficiency (that is, profit over weight).

– **Constraint Programming Approach (CPA):** A variant of PA where, on top of pruning the search tree by means of the simple linear relaxation, we also perform knapsack constraint filtering and propagation at every search node.

– **Full Inference Approach (FIA):** The combination of VIA and CPA.

3 Two Performance Measures for Value Selection Heuristics

When evaluating value selection heuristics, the question arises how we should measure their quality. As mentioned in the introduction, for satisfiability problems theoretical models have been considered where assumptions were made regarding the probability with which a heuristic would guide us to a feasible solution. For optimization problems like knapsack, the mere existence of a feasible solution is not interesting. What actually matters is the quality of the best solution that can be found in the subtree that the heuristic tells us should be considered first.

This leads us directly to the first performance measure that we will study empirically in our experiments. The *probability measure* assesses the probability that the heuristic steers us in the direction of the subtree which contains the better solution, whenever that choice matters. That is to say that we consider only those cases where making a correct heuristic decision makes a difference. This is the case when at least one subtree contains an improving solution, and when the quality of the best solutions in both subtrees is different. To see how this relevance probability evolves in practice, in our experiments we keep track of how many search nodes are being considered at every depth level, and for how many of those choice points the heuristic decision to favor one child over the other makes a difference.

The second measure that we consider is the *accuracy measure* which takes a quantitative view of the heuristic choices by comparing the actual solution quality of the best solutions in both subtrees. Rather than measuring the actual difference in objective value (which would artificially increase the impact of instances with larger optimal solutions), we measure the difference relative to the current gap between upper and lower bound. Therefore, when we find that the average accuracy of a value selection heuristic at some depth is 40%, then this means that, by following the heuristic, we are guided to a subtree whose best solution closes on average 40% more of the current gap than the best solution in the subtree whose investigation was postponed.

4 Heuristic Accuracy as a Function of Depth

We specified the approaches that we use to solve different benchmark classes of knapsack instances, we introduced several value selection heuristics for the problem, and we described two different performance measures for these heuristics. In this section, we
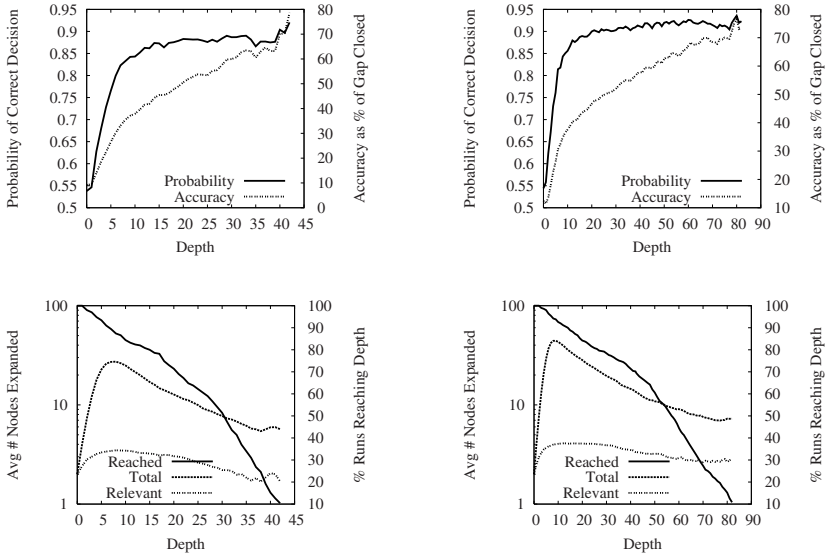
Fig. 3. Top: Average development of the probability and accuracy measure over depth for the rounding heuristic on 500 uncorrelated knapsack instances with 50 items (left) or 100 items (right). The solid line ('Probability') goes with the left vertical axis and depicts the probability that the heuristic makes the correct choice. The dashed line ('Accuracy') goes with the right vertical axis and depicts the average gap-closing percentile by which the heuristic choices outperform their opposites. Bottom: On the same benchmark sets, we show the percentage of instances that reach a certain depth (solid line, right vertical axis, 'Reached'). The dashed and dotted lines go with the left vertical axis (log-scale) and measure the average number of search nodes ('Total') on each depth level and the average number of nodes for which the heuristic choice is relevant ('Relevant'), respectively.

now investigate how the different heuristics perform in comparison. We implemented the algorithms presented earlier in Java 5, and we used Ilog Cplex 10.1 to compute linear continuous relaxations in algorithms VIA and FIA. Moreover, we implemented our own constraint propagation engine for algorithms CPA and FIA.

Note that our code is not tuned for speed as it has to gather and maintain all kinds of statistical information during search anyway. Fortunately, computational efficiency does not influence our performance measures for search heuristics or how that performance depends on the current state of the search or the inference mechanisms that are employed. Furthermore, within the branch-and-bound framework that we consider, the different knapsack heuristics all cause comparable overhead which is why a count of the number of search nodes gives us a very good estimate of the total time needed as well. However, note that this argument does not hold for the different inference methods that we will employ in Section 5.

To get a first insight into typical heuristic behavior, in Figure 3 we assess accuracy and probability of the rounding heuristic (that performs well with DFS and LDS, see Figures 1 and 2) when we run our baseline algorithm PA on uncorrelated instances. We see that the heuristic performs really well: Although it is rather uninformed at the
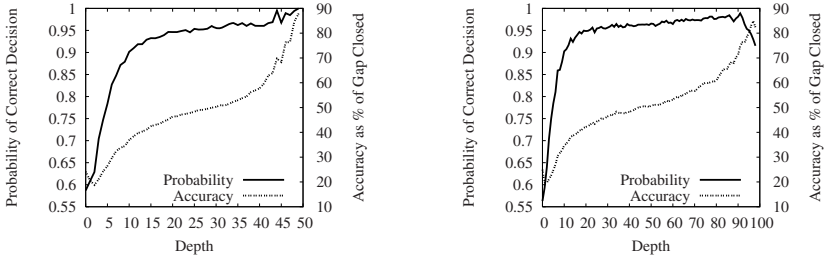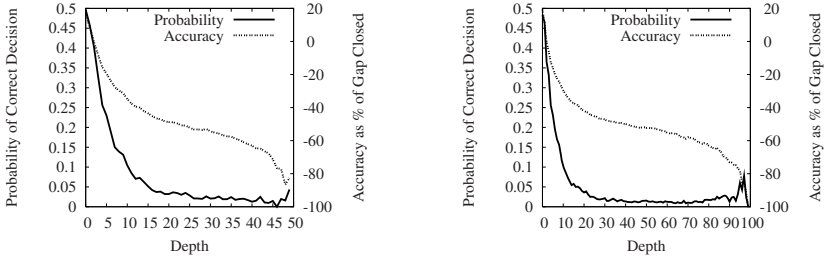
Fig. 4. The average probability and accuracy of the best-first heuristic on 500 weakly correlated knapsack instances with 50 (left) and 100 items (right)

beginning, it quickly becomes better and better. At depth 10, it already favors branches that close, on average, about 40% more of the current gap between upper and lower bound, and it steers us in the right direction in about 87% of the relevant cases.

Moreover, we observe a very distinct behavior with respect to the probability that the rounding heuristic makes the right choice when we ignore the extent to which the choice is better: The probability of favoring the subtree that contains the better solution is almost random (close to one half) at the very beginning of the search. Then, the probability of moving to the better subtree grows very quickly until it levels out at about 89% roughly at depth level 15. From then on, the probability stays pretty much constant. (We observe another brief increase to almost perfect prediction when we are very close to the maximum depth in the tree, but given the few runs that reach this depth and the few search-nodes on this depth this is not statistically significant.) This description holds for both the 50 and the 100 items benchmark, whereby for the latter the probability levels out only slightly later. Although we cannot show all our data here, we observe this *same* basic curve on the other classes of knapsack instances (including the one considered in Figures 1 and 2) and also for other "good" heuristics like best-first or exclusion. To give two more examples, in Figures 4 and 7, we depict the best-first and the exclusion heuristics, this time on weakly correlated knapsack instances.

To draw a first conclusion, our data suggests that a good value selection heuristic for knapsack has a much larger probability of misleading us higher up in the tree. Moreover, good heuristics become more accurate quickly and then keep their performance pretty much constant. Note that this behavior is addressed by neither LDS nor DDS: LDS has only a slight tendency to reconsider discrepancies higher up a little bit earlier than further down below, but only as far as leaves with the same number of discrepancies are concerned. Consequently, it wastes a lot of time reconsidering heuristic choices that are made deep down in the tree which are probably quite accurate, instead of putting more effort into the investigation of the more questionable decisions. DDS, on the other hand, does very well by quickly reconsidering choices that were made early in the search. However, it is not justified to assume that perfect predictions are already achieved at some rather shallow search-depth. From a certain depth-level on, simply minimizing the total number of discrepancies appears to be much better.

In the bottom part of Figure 3, we can see the number of choice points where the heuristic choice makes a difference is an order of magnitude lower than the total number of search nodes. This is caused by the fact that a good part of our search consists in a
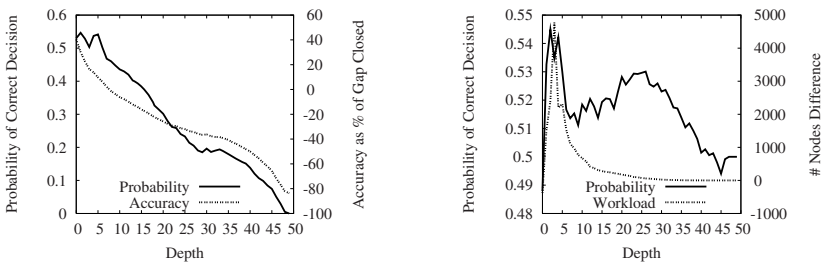
Fig. 5. The average probability and accuracy of the momentum heuristic on 500 strongly correlated knapsack instances with 50 (left) and 100 items (right)

proof of optimality. In this part of the search, the value selection heuristic is immaterial, just as it would be when considering an unsatisfiable instance.

It is interesting to note that, once the search has reached a sufficient depth, the probability that good value selection is relevant at some choice point is more or less independent of the depth level. We observe that the distance between the line depicting the average total number of choice points per level and the number of relevant choice points is almost constant. As we use a logarithmic scale, this indicates that the ratio of the number of relevant nodes over the total number of nodes does not change very much from some depth level on. In [12], the assumption is made that the relevance of value selection was constant. As with the accuracy of good heuristic value selection, this assumption appears only valid after we reached some critical depth in the tree. On very high levels, making good choices is much more of a gamble and at the same time much more likely to be important. Both are indications that a good search strategy ought to reconsider very early search decisions much more quickly than LDS does.

Thus far we have only studied heuristics that perform well. In Figure 5, we depict the performance of the momentum heuristic on strongly correlated instances. We see clearly that the heuristic performs poorly. On second thought, this is hardly surprising as it has a strong tendency to exclude high efficiency items and to include lower efficiency items. What our graph shows, however, is that making the general assumption that all heuristics ought to perform better as we dive deeper into the tree is quite wrong. As a matter of fact, for bad heuristics, performance may even decay.

Fig. 6. Accuracy and probability measure for the inclusion heuristic on 500 almost strongly correlated knapsack instances with 50 items each
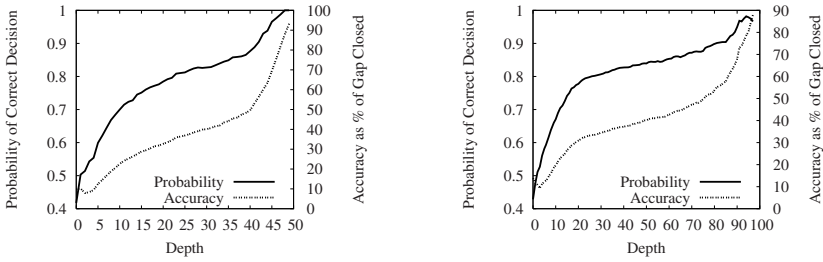
Fig. 7. The average probability and accuracy of the best-first (top) and the exclusion heuristic (bottom) on 1000 weakly correlated knapsack instances with 50 (left) and 100 items (right)

Now, what about the inclusion heuristic? We plot its performance in Figure 6. We see that the inclusion heuristic makes increasingly worse predictions where better solutions can be found as we dive deeper into the tree. This explains, of course, why inclusion is bound to perform poorly in combination with LDS as we saw in Figure 2. But why did it do so well in combination with DFS (see Figure 1)? The answer to this riddle is found in the right plot in Figure 6: Here, we assess the quality of inclusion while performing DFS by comparing the workload (that is, the total number of nodes) when exploring the subtree the heuristic chooses first. Again, we show the quantitative dependency of depth as the average number of nodes that the heuristic needs to investigate less (see the curve denoted "workload"), and a qualitative comparison which shows how often the heuristic investigates the subtrees in the right order (see the curve denoted "probability"). We see that, by including items first, the inclusion heuristic guides the search to more shallow subtrees first as the inclusion of items obviously has to be paid for by a reduction in remaining capacity. In those shallow subtrees, we quickly find a no-good (a new lower bound) which can be exploited when searching the second branch where better solutions can be found more quickly. When combining inclusion with LDS, however, we never get to searching those other branches, and while no-goods are helpful, we cannot afford to pay for them by excluding from our search the most interesting parts of the search space.

This example shows impressively that the accuracy of heuristic predictions can be abysmal even when a heuristic works well when combined with one specific search strategy (compare with Figure 1). In particular, even when a heuristic performs well with one search strategy, this is no indication that it is also suited *to guide* an incomplete search that is terminated early under hard time constraints. If one is to avoid that a heuristic like inclusion is combined with LDS to form an algorithm that is expected to provide good solutions quickly, then there is but one alternative: The performance of heuristics must be studied much more carefully than by a simple evaluation based on a global performance measure like best quality after some time-limit within a spe-cific approach. Instead, a *direct assessment* of the accuracy of heuristic decisions as we provide it here is necessary. Otherwise, when designing algorithms, we are bound to a trial-and-error methodology that is far from the standards of scientific engineering.

Our direct assessment of the accuracy of heuristic decision making has allowed us to identify that there exists an interplay of heuristic decisions and no-goods that are learned during search – a matter which we have never seen discussed when heuristics for a problem are designed. What is more, we have shown that the belief that heuristics would generally become more accurate with increasing depth is wrong. This raises the

question why good heuristics *are* actually getting more accurate! An important insight here is that a heuristic like exclusion, which plainly sets the critical item to zero first, is of course not getting any "smarter" deeper down in the tree. Compare the accuracy at depth level 50 of the 100 item benchmark and the accuracy at level 0 in the 50 item benchmark in Figure 7. Actually, one should expect the accuracies to be the same. The reason why they differ is of course not that exclusion got more "experienced" while branching to depth-level 50 in the 100 item instances. The reason why the accuracy is higher than at the root-node level in the 50 item case can only lie in the different distribution of instances where the heuristic choices matter at depth-level 50 in the 100 item case. That is, exclusion (and the other heuristics for that matter) is getting more and more accurate because it is in some way self-enforcing: the decisions made higher up in the search tree result in problems for which the heuristic is more accurate. Inclusion, on the other hand, is not a good guide where good solutions will be found because it is self-weakening (which becomes clear when considering the repeated use of the inclusion heuristic which always trades the higher efficiency of an item to the left of the critical item for the efficiency of the latter).

We are not aware that self-enforcement has been noted before as an important aspect when designing value-selection heuristics. There exists, however, some work that tries to integrate different heuristics rather than sticking slavishly to the same one. For example, in [1], Balas et al. found that randomly combining different heuristics at each step of a greedy algorithm for set-covering actually produces better solutions than sticking to the single best heuristic for each instance throughout the algorithm.

## 5   How Inference Affects the Robustness of Search Heuristics

The final aspect that we are interested in is to what extent inference methods can influence the accuracy and the relevance of search heuristics. In this section, we therefore show how search heuristics perform when used with our different algorithm variants PA, VIA, CPA, and FIA (see Section 2).

Consider Figure 8, where we compare the accuracy of the rounding heuristic when used in combination with inference mechanisms of different strengths. On the left we consider algorithms PA and CPA that do not use knapsack cuts. The latter are used for the plots on the right that show the performance of rounding in combination with VIA and FIA. Analogously, at the top we show algorithms PA and VIA that do not use constraint filtering, whereas at the bottom we consider algorithms CPA and FIA that both propagate knapsack constraints. The results look very counter-intuitive: The more inference we perform, the worse the heuristic predictions become!

However, a look at the corresponding relevance data in Figure 9 reveals that, in reality, inference makes heuristic predictions far more robust: By adding just knapsack cuts, we already prevent searches from exploring deeper nodes. It is very impressive how constraint propagation (both in combination with knapsack cuts or alone) boosts this trend dramatically: Without constraint filtering, 50% of all searches reach a depth of 85, while with knapsack constraints, they only reach a depth of 20. On top of that, the maximum average number of relevant nodes per depth level is reduced to only 2.7![1]

---

[1] Note that this drastic reduction is also the reason why the curves in Figure 8 make such a ragged impression. There are simply far fewer sample points to average over.
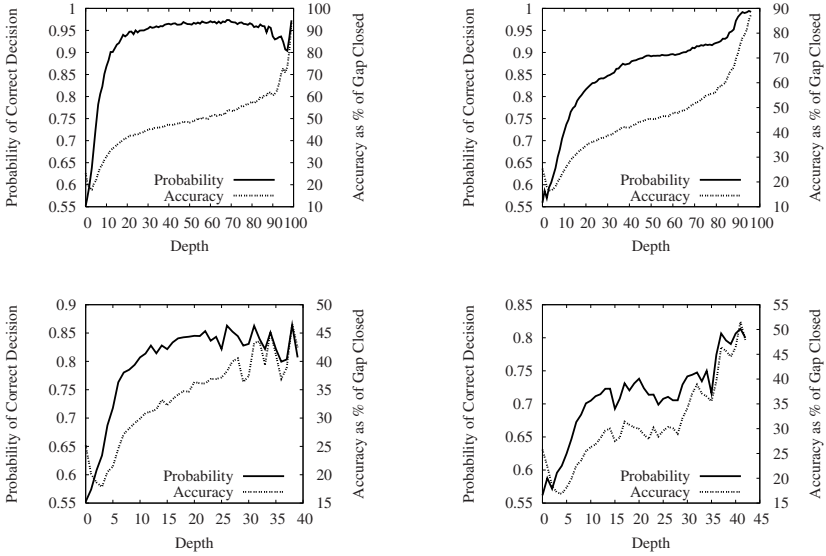
Fig. 8. Accuracy and probability of the rounding heuristic with algorithms PA (top left), VIA (top right), CPA (bottom left), and FIA (bottom right) on 1000 weakly correlated knapsack instances with 100 items. The 'Probability' of making the right decision goes with the left axis, the actual 'Accuracy' is again measured on the right axis as the relative gain in terms of percent of gap closed.
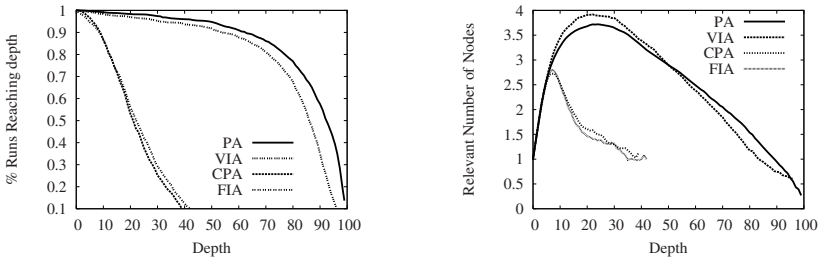
Fig. 9. Relevance of the rounding heuristic when employed within algorithms PA, VIA, CPA, and FIA on 1000 weakly correlated knapsack instances with 100 items. The left plot shows the percentage of instances that reach a certain depth, the right how many nodes are relevant, as an average over all runs that reach that depth.

Note that stronger inequalities like Mixed-Integer Gomory Cuts may have a similarly strong effect. However, while our statistics recording code does not allow us to provide a quantitative comparison, we would like to note that we found that constraint propagation could be conducted much faster than the computation of the strengthened linear relaxation bounds.

We conclude that stronger inference techniques can reduce the importance of good heuristic predictions. As a consequence, value selection heuristics are left with less room to make a dramatic difference and therefore appear less accurate on average.

6   Conclusions

We conducted an empirical study regarding the accuracy of value selection heuristics within incomplete systematic search. We found that global performance measures do not enable us to deduce how adequate heuristic predictions are. By measuring heuristic accuracy as a function of depth, we found that good value selection functions are most error prone when only few branching decisions have been made while the expected relevance of these decisions is greater. However, while bad heuristics may even decay with depth, good knapsack heuristics quickly improve their performance, which then stays practically constant for the remainder of the search.

To devise better value heuristics, we found two aspects to be essential: First, heuristics that quickly generate high-quality no-goods can actually work faster than more accurate heuristics, depending on the search strategy within which they are employed (recall the good performance of the inclusion heuristic within DFS). Second, improved heuristic accuracy is the result of self-enforcement: decisions higher-up in the tree drive the distribution of nodes where the heuristic choice matters in such a way that for these nodes the heuristic works better. In order to devise superior search heuristics, we believe that both aspects deserve to be studied in much more detail.

Finally, we found that inference methods render heuristic decisions far less relevant and thereby improve the robustness of search. Of course, whether invoking expensive inference techniques pays off or not depends largely on the accuracy of the heuristics that are available: After all, a good and robust value selection heuristic has the potential to guide us to feasible or optimal solutions, even when we choose branching variables poorly, or when inference is limited.

With respect to future work, we need to investigate whether the characteristic curve that describes the accuracy of many good heuristics for knapsack instances of various benchmark classes also applies to good heuristics of other combinatorial problems. Based on our findings, we then intend to devise new search strategies that actively incorporate the characteristic evolution of heuristic accuracy during search.

References

1. Balas, E., Carrera, M.: A dynamic subgradient-based branch-and-bound procedure for set covering. Operations Research 44, 875–890 (1996)
2. Beacham, A., Chen, X., Sillito, J., van Beek, P.: Constraint Programming Lessons Learned from Crossword Puzzles. In: Canadian Conference on AI, pp. 78–87 (2001)
3. Cooper, M.C.: An Optimal k-Consistency Algorithm. AI 41, 89–95 (1989)
4. Crowder, H., Johnson, E., Padberg, M.: Solving large scale zero-one linear programming problem. Operations Research 31, 803–834 (1983)
5. Dantzig, G.: Discrete variable extremum problems. Operations Research 5, 226–277 (1957)
6. Fahle, T., Sellman, M.: Cost-Based Filtering for the Constrained Knapsack Problem. AOR 115, 73–93 (2002)
7. Focacci, F., Lodi, A., Milano, M.: Cutting Planes in Constraint Programming. In: CP-AI-OR, pp. 45–51 (2000)
8. Freuder, E.: Backtrack-Free and Backtrack-Bounded Search. In: Search in Artificial Intelligence, pp. 343–369 (1988)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability (1979)

10. Gomes, C., van Hoeve, W., Leahu, L.: The Power of Semidefinite Programming Relaxations for MAXSAT. In: Beck, J.C., Smith, B.M. (eds.) CPAIOR 2006. LNCS, vol. 3990, Springer, Heidelberg (2006)
11. Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. American Mathematical Society 64, 275–278 (1958)
12. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. IJCAI, 607–613 (1997)
13. Hooker, J.N.: A search-infer-and-relax framework for integrating solution methods. In: CPAIOR, pp. 243–257 (2005)
14. Martello, S., Pisinger, D., Toth, P.: Dynamic programming and tight bounds for the 0-1 knapsack problem. Management Science 45, 414–424 (1999)
15. Pisinger, D.: Where are the hard knapsack problems? Computers and Operations Research 32(9), 2271–2284 (2005)
16. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: CABOP: A Fast Optimal Algorithm for Winner Determination in Combinatorial Auctions. Management Science 51(3), 374–390 (2005)
17. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach (2002)
18. Trick, M.: A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints. In: CP-AI-OR, pp. 113–124 (2001)
19. Walsh, T.: Depth-bounded discrepancy search. In: IJCAI, pp. 1388–1393 (1997)
20. Williams, R., Gomes, C., Selman, B.: On the Connections between Heavy-tails, Backdoors, and Restarts in Combinatorial search. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, Springer, Heidelberg (2004)