Backdoors to Combinatorial Optimization: Feasibility and Optimality

Bistra Dilkina[1], Carla P. Gomes[1], Yuri Malitsky[2], Ashish Sabharwal[1], and Meinolf Sellmann[2]

[1] Department of Computer Science, Cornell University, Ithaca, NY 14853, U.S.A.
{bistra,gomes,sabhar}@cs.cornell.edu
[2] Department of Computer Science, Brown University, Providence, RI 02912, U.S.A.
{ynm,sello}@cs.brown.edu

Abstract. There has been considerable interest in the identification of structural properties of combinatorial problems that lead, directly or indirectly, to the development of efficient algorithms for solving them. One such concept is that of a backdoor set—a set of variables such that once they are instantiated, the remaining problem simplifies to a tractable form. While backdoor sets were originally defined to capture structure in decision problems with discrete variables, here we introduce a notion of backdoors that captures structure in optimization problems, which often have both discrete and continuous variables. We show that finding a feasible solution and proving optimality are characterized by backdoors of different kinds and size. Surprisingly, in certain mixed integer programming problems, proving optimality involves a smaller backdoor set than finding the optimal solution. We also show extensive results on the number of backdoors of various sizes in optimization problems. Overall, this work demonstrates that backdoors, appropriately generalized, are also effective in capturing problem structure in optimization problems.

Keywords: search, variable selection, backdoor sets.

1 Introduction

Research in constraint satisfaction problems, in particular Boolean satisfiability (SAT), and in combinatorial optimization problems, in particular mixed integer programming (MIP), has had many historic similarities (see, e.g., [2]). For example, the earliest solution methods for both started out as processes that non-deterministically or heuristically chose new inferred information to add repeatedly until the problem was fully solved. In SAT, this took the form of adding "resolvents" of two clauses and formed the original Davis-Putnam procedure. In MIP, this took the form of repeatedly adding cuts. In both cases, it was soon observed that the vast array of possibilities for such resolvents and cuts to add can easily turn into a process without much focus, and thus with limited success. The remedy seemed to be to apply a different, top-down technique instead of deriving and adding new information bottom-up. The top-down process took the form of DPLL search for SAT and of branch-and-bound for MIP. Again, it was realized that such branch-and-bound style systematic search has its own drawbacks, one of them being not *learning* anything as the search progresses. The fix—a relatively recent development in the long history of SAT and MIP methods—was to combine the two approaches. In SAT, this took the form of "clause learning" during the branch-and bound process, where new derived constraints are added to the problem upon backtracking. In MIP, this took the form of adding "cuts" and "tightening bounds" when exploring various branches during the branch-and-bound search.

This similarity between SAT and MIP research suggests that concepts that have been used successfully in one realm can perhaps also be extended to the other realm and lead to new insights. We investigate this from the angle of applying ideas from SAT to MIP. In particular, we consider heavy-tailed behavior of runtime distribution and the related concept of backdoor sets. It has been observed that (randomized) SAT solvers often exhibit a large variation in runtimes even when using randomization only for tie-breaking. At the same time, one often sees a SAT solver solve a hard real-world problem very quickly when in fact the problem should have been completely out of the reach of the solver by standard complexity arguments. Backdoor sets provide a way to understand such extremely short runs often seen on structured real-world instances and rarely seen on randomly generated instances.

We remark that the study of backdoors in constraint satisfaction problems was motivated by the observation that the performance of backtrack-style search methods can vary dramatically depending on the order of variable and value selection during the search. In particular, backtrack search methods exhibit large variance in runtime for different heuristics, different problem instances, and, for randomized methods, for different random seeds even on the same instance. The discovery of the "heavy-tailed" nature of the runtime distributions in the context of SAT [4, 5, 7, 10] has resulted in the effective introduction of randomization and restart techniques [6] and has been related to the presence of small backdoors [12]. A question, then, naturally arises: *do the runtime distributions of combinatorial optimization problems also exhibit a similar behavior? In particular, are these distributions heavy-tailed?*

Formally, heavy-tail distributions exhibit power-law decay near the tail end of the distribution and are characterized by infinite moments. The distribution tails are asymptotically of the Pareto-Levy form. Most importantly for us, the log-log plot of the tail of the survival function (i.e., how many instances are *not* solved in a given runtime) of a heavy-tailed distribution exhibits *linear behavior*. We considered the runtime distributions of MIP instances from the MIPLIB library [1], using CPLEX's [8] branch-and-bound search with a randomized branching heuristic. While heavy-tailed behavior has been reported mostly in the context of constraint satisfaction, some of the MIP optimization instances in our experiments did show heavy-tailed behavior.

These observations for MIP optimization problems motivate an in-depth study of the concept of backdoors for these problems, which is the main focus of this paper. Informally, backdoors for constraint satisfaction are sets of variables the systematic search can, at least in principle, be limited to when finding a solution or proving infeasibility. We extend the concept of backdoor sets to optimization problems, which raises interesting new issues not addressed by earlier work on backdoor sets for constraint satisfaction. We introduce "weak optimality backdoors" for finding optimal solutions and "optimality-proof backdoors" for proving optimality. The nature of optimization algorithms, often involving adding new information such as cuts and tightened bounds as the search progresses, naturally leads to the concept of "order-sensitive" backdoors, where information learned from previous search branches is allowed to be used by the sub-solver underlying the backdoor. This often leads to much smaller backdoors than the "traditional" ones.

We investigate whether significantly small backdoors also exist for standard benchmark instances of mixed integer programming optimization problems, and find that such instances often have backdoors involving under 5% of the discrete variables. Interestingly, sometimes the optimality-proof backdoors can in fact be smaller than the weak optimality backdoors, and this aligns with the relative runtime distributions for these problems when finding an optimal solution vs. when proving optimality. A large part of our experimental work involves the problem of determining how many backdoors of various kinds and sizes exist in such optimization problems, and whether information provided by linear programming relaxations (e.g., the "fractionality" of the variables in the root LP relaxation) can be used effectively when searching for small backdoors. Our results provide positive answers to these questions.

2 Background: Backdoors for Constraint Satisfaction

We begin by recalling the concept of weak and strong backdoor sets for constraint satisfaction problems. For simplicity of exposition, we will work with the Boolean satisfiability (SAT) problem in this section, although the concepts discussed apply equally well to any discrete constraint satisfaction problem.

Backdoor sets are defined with respect to efficient sub-algorithms, called *sub-solvers*, employed within the systematic search framework of SAT solvers. In practice, these sub-solvers often take the form of efficient procedures such as unit propagation, pure literal elimination, and failed-literal probing. In some theoretical studies, solution methods for structural sub-classes of SAT such as 2-SAT, Horn-SAT, and RenamableHorn-SAT have also been studied as sub-solvers. Formally [11], a *sub-solver $\mathcal{A}$ for SAT* is any poly-time algorithm satisfying certain natural properties on every input formula F: (1) Trichotomy: $\mathcal{A}$ either determines F correctly (as satisfiable or unsatisfiable) or fails; (2) $\mathcal{A}$ determines F for sure if F has no constraints or an already violated constraint; and (3) if $\mathcal{A}$ determines F, then $\mathcal{A}$ also determines $F|_{x=0}$ and $F|_{x=1}$ for any variable x.

In the definitions of backdoor sets that follow, the sub-solver $\mathcal{A}$ will be implicit. For a formula F and a truth assignment τ to a subset of the variables of F, we will use $F[\tau]$ to denote the simplified formula obtained after applying the (partial) truth assignment to the affected variables.

Definition 1 (Weak and Strong Backdoors for SAT [11]). *Given a Boolean formula F on variables X, a subset of variables $B \subseteq X$ is a* weak backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if for* some *truth assignment $\tau : B \to \{0,1\}$, $\mathcal{A}$ returns a satisfying assignment for $F[\tau]$. Such a subset B is a* strong backdoor *if for* every *truth assignment $\tau : B \to \{0,1\}$, $\mathcal{A}$ returns a satisfying assignment for $F[\tau]$ or concludes that $F[\tau]$ is unsatisfiable.*

Weak backdoor sets capture the fact that a well-designed heuristic can get "lucky" and find the solution to a hard satisfiable instance if the heuristic guidance is correct even on the small fraction of variables that constitute the backdoor set. Similarly, strong backdoor sets B capture the fact that a systematic tree search procedure (such as DPLL) restricted to branching only on variables in B will successfully solve the problem, whether satisfiable or unsatisfiable. Furthermore, in this case, the tree search procedure restricted to B will succeed independently of the order in which it explores the search tree.

3 Backdoor Sets for Optimization Problems

This section extends the notion of backdoor sets from constraint satisfaction problems to combinatorial optimization problems. We begin by formally defining optimization problems and discussing desirable properties of sub-solvers for such problems. Without loss of generality, we will assume throughout this text that the optimization to be performed is minimization. For simplicity of notation, we will also assume that all variables involved have the same value domain, D.

Definition 2 (Combinatorial Optimization Problem). *A combinatorial optimization problem is a four-tuple (X, D, C, z) where $X = \{x_i\}$ is a set of variables with domain D, C is a set of constraints defined over subsets of X, and $z : D^{|X|} \to \mathbb{Q}$ is an objective function to be minimized.*

A constraint $c \in C$ over variables $var(c)$ is simply a subset of all possible value assignments to the variables involved in c, i.e., $c \subseteq D^{|var(c)|}$. A value assignment v is said to *satisfy* c if the restriction of v to the variables $var(c)$ belongs to the set of value tuples constituting c.

Definition 3 (Sub-Solver for Optimization). *A* sub-solver *$\mathcal{A}$ for optimization is an algorithm that given as input a combinatorial optimization problem (X, D, C, z) satisfies the following four conditions:*
[(a)]

1. Trichotomy: *$\mathcal{A}$ either infers a lower bound on the optimal objective value z or correctly determines (X, D, C, z) (as either unsatisfiable or as optimized providing a feasible solution that is locally optimal),*
2. Efficiency: *$\mathcal{A}$ runs in polynomial time,*
3. Trivial solvability: *$\mathcal{A}$ can determine whether (X, D, C, z) is trivially satisfied (has no constraints) or trivially unsatisfiable (has an empty constraint), and*

4. Self-reducibility: *If $\mathcal{A}$ determines (X, D, C, z), then for any variable x_i and value $v \in D$, $\mathcal{A}$ also determines $(X, D, C \cup \{x_i = v\}, z)$.*

For some partial assignments, the sub-solver might learn a new lower bound on the objective value. For some partial assignments, the solver may find a feasible solution that is a locally optimal solution. Any feasible solution provides an upper bound on the optimal objective value. Hence, for some partial assignments, the sub-solver might learn a new upper bound on the objective value.

In extending the notion of backdoor sets to optimization problems, we need to take into account that we face two tasks in constrained optimization: first, we need to find a feasible and optimal solution, and second, we need to prove its optimality which essentially involves proving infeasibility of the problem when the objective bound is reduced beyond the optimal value. This naturally leads to three kinds of backdoors: *weak optimality backdoors* will capture the task of finding optimal solutions, *optimality-proof backdoors* will capture the task of proving optimality given the optimal objective value, and *strong optimality backdoors* will capture the full optimization task, i.e., finding an optimal solution and proving its optimality.

Weak backdoors for optimization are the most straightforward generalization from the constraint satisfaction realm. One notable difference, however, is that since we are trying to decouple the solution-finding task from the optimality-proof task, we assume that the solution-finding task is, in a sense, somehow aware of the optimal objective value and can stop when it hits an optimal solution. In our experiments designed to identify backdoor sets, we achieve this by pre-computing the optimal objective value and forcing the search to stop when a feasible solution achieving this objective value is encountered.

We use the word "traditional" in the next few definitions to distinguish them from the concept of order-sensitive backdoors to be discussed in Section 3.1. In the following, $C \cup \tau$ denotes adding to C the constraint $\{(v_1, \ldots, v_n) \mid \forall x_i \in B,\ v_i = \tau(x_i)\}$ imposing the partial assignment τ on the variables in B.

Definition 4 ((Traditional) Weak Optimality Backdoor). *Given a combinatorial optimization problem (X, D, C, z), a subset of the variables $B \subseteq X$ is a* (traditional) weak optimality backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if there exists an assignment $\tau : B \rightarrow D$ such that $\mathcal{A}$ returns a feasible solution for $(X, D, C \cup \tau, z)$ which is of optimal quality for (X, D, C, z).*

In contrast to decision problems, solving an optimization instance also requires proving that no better feasible solution exists. Therefore, we define the notion of backdoor sets for the optimality proof itself. Once we have found an optimal feasible solution x^*, this immediately also provides the optimal upper bound to the objective, making the new problem of seeking a better objective value infeasible. Optimality-proof backdoors are sets of variables that help one deduce this infeasibility efficiently.

Definition 5 ((Traditional) Optimality-Proof Backdoor). *Given a combinatorial optimization problem (X, D, C, z) and an upper bound z^* on the objective value, a subset of the variables $B \subseteq X$ is a* (traditional) optimality-proof backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if for every assignment $\tau : B \to D$, $\mathcal{A}$ correctly decides $(X, D, C \cup \tau \cup \{z < z^*\}, z)$ to be infeasible.*

The notion of optimality-proof backdoor allows us to decouple the process of finding feasible solutions from proving the optimality of a bound. An optimality-proof backdoor is particularly relevant when there is an external procedure that finds good feasible solutions (e.g., heuristic greedy search). Given the best solution quality found by the greedy search, we can use an optimality-proof backdoor to confirm that no better solution exists or perhaps to disprove the bound by finding a better feasible solution.

Both the definition of weak optimality backdoor and of optimality-proof backdoor implicitly or explicitly rely on the knowledge of an upper bound z^* on the objective value, i.e., they do not capture solving the original optimization problem for which the optimal value is unknown. Recall that strong backdoors for constraint satisfaction problems capture the set of variables that are enough to fully solve the problem—either prove its infeasibility or find a solution. We would like to define a similar notion for optimization problems as well. To this end, we define strong backdoors for optimization, which are enough to both find an optimal solution and prove its optimality, or to show that the problem is infeasible altogether. When the problem is feasible, a strong backdoor set is both a weak optimality backdoor and an optimality-proof backdoor.

Definition 6 ((Traditional) Strong Optimality Backdoor). *Given a combinatorial optimization problem (X, D, C, z), a subset of the variables $B \subseteq X$ is a* (traditional) strong optimality backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if it satisfies the following conditions. For every assignment $\tau : B \to D$, $\mathcal{A}$ infers a lower bound $lb(\tau)$ on the optimal objective value for $(X, D, C \cup \tau, z)$; $lb(\tau) = +\inf$ if infeasible. If, for τ, the sub-solver also finds an optimal solution $\hat{x}(\tau)$ for $(X, D, C \cup \tau, z)$, then let $\hat{z}(\tau) = z(\hat{x}(\tau))$, else let $\hat{z}(\tau) = +\inf$. We must have:* $\min_\tau lb(\tau) = \min_\tau \hat{z}(\tau)$.

3.1 Order-Sensitive Backdoors

We now discuss an issue that arises naturally when we work with backdoor sets for state-of-the-art optimization algorithms, such as CPLEX for mixed integer programming (MIP) problems: *order-sensitivity* of backdoor sets. Order-sensitivity plays an increasingly important role as we extend the notion of backdoors to constraint optimization problems.

The standard requirement implicit in the notion of backdoor sets in constraint satisfaction problems is that the underlying systematic search procedure restricted to backdoor variables should succeed independently of the order in which it explores various truth valuations of the variables; in fact, for strong backdoors, the sub-solver must succeed on every single search branch based solely on the value assignment to the backdoor variables. This condition, however, ignores an important fact: a crucial feature of most branch-and-bound algorithms for constrained optimization problems is that they *learn* information about the search space as they explore the search tree. For example, they learn new bounds on the objective value and the variables, and they might add various kind of "cuts" that reduce the search space without removing any optimal solution. These tightened bounds and cuts potentially allow the sub-solver to later make stronger inferences from the same partial assignment which would have normally not lead to any strong conclusions. Indeed, in our experiments designed to identify weak optimality backdoor sets for MIP problems, it was often found that variable-value assignments at the time CPLEX finds an optimal solution during search do not necessarily act as traditional weak backdoors, i.e., feeding back the specific variable-value assignment doesn't necessarily make the underlying sub-solver find an optimal solution. This leads to a natural distinction between "traditional" (as defined above) and "order-sensitive" weak optimality backdoors. In the following definitions, *search order* refers to the sequence of branching decisions that a search method uses in exploring the search space and possibly transferring any available learned information (such as cuts or tigher bounds) from previously explored branches to subsequent branches.

Definition 7 (Order-Sensitive Weak Optimality Backdoor). *Given a combinatorial optimization problem (X, D, C, z), a subset of the variables $B \subseteq X$ is an* order-sensitive weak optimality backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if there exists some search order involving only the variables in B that leads to an assignment $\tau : B \to D$ such that $\mathcal{A}$ returns a feasible solution for $(X, D, C \cup \tau, z)$ which is of optimal quality for (X, D, C, z).*

In fact, added cuts and tightened bounds form an integral part of solving a MIP optimization problem and can critically help even when "only" detecting a feasible solution of optimal quality. The same distinction also applies to optimality-proof backdoors and to strong backdoors, simplifying the rather cumbersome definition in the latter case.

Definition 8 (Order-Sensitive Optimality-Proof Backdoor). *Given a combinatorial optimization problem (X, D, C, z) and an upper bound z^* on the objective value, a subset of the variables $B \subseteq X$ is an* order-sensitive optimality-proof backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if there exists some search order involving only the variables in B such that $\mathcal{A}$ correctly decides $(X, D, C \cup \{z < z^*\}, z)$ to be infeasible.*

Definition 9 (Order-Sensitive Strong Optimality Backdoor). *Given a combinatorial optimization problem (X, D, C, z), a subset of the variables $B \subseteq X$ is an* order-sensitive strong backdoor *(w.r.t. a specified sub-solver $\mathcal{A}$) if there exists some search order involving only the variables in B such that $\mathcal{A}$ either correctly decides that the problem is infeasible, or finds an optimal solution and proves its optimality.*

4 Experimental Evaluation

To investigate the distribution of backdoor sizes in optimization problems, we consider the domain of Mixed Integer Programming. In our empirical study, we use instances from the MIPLIB library [1], and employ the branch-and-bound search framework provided by CPLEX [8]. Due to the computationally intensive analysis performed in this study, we only evaluate MIPLIB instances that could be solved reasonably fast with CPLEX.

The sub-solver applied by CPLEX at each search node of the branch-and-bound routine uses a dual simplex LP algorithm in conjunction with a variety of cuts. In our previous study [3] of backdoors in Satisfiability problems, we investigated the sub-solver routine used in *Satz* [9] which applied probing to each search node. Similarly here, we set CPLEX to use strong branching, adding a lot of additional inference at each node. In summary, the sub-solver is *dual simplex+CUTS+probing.*[1]

We investigate the probability that a randomly selected subset of the variables of a given cardinality k is a backdoor. To approximate this probability, we sample many sets (500) of each given size, and for each evaluate whether the chosen set is a traditional weak optimality backdoor, order-sensitive weak optimality backdoor, and/or optimality-proof backdoor.

In our experiments we consider order-sensitive optimality-proof backdoors (and not traditional optimality-proof backdoors). For brevity, we will refer to them simply as optimality-proof backdoors. To decide whether a given set B of variables is an optimality-proof backdoor, we initialize CPLEX with the optimal solution and allow branching only on the set B. As soon as we reach a search node at which all variables of B are fixed but the infeasibility of the sub-problem at the node cannot be concluded, we reject B. Note that with a different search order, CPLEX could have succeeded in proving infeasibility if the alternative order provided stronger cuts earlier. Hence our results provide a lower bound on the probability that a set of a certain size is an optimality-proof backdoor.

To decide whether a given set B of variables is an order-sensitive weak optimality backdoor, we allow branching only on the chosen set. As soon as we find an incumbent which has optimal objective value, (precomputed ahead of time), we accept the set. That is, we stop the search when an optimal solution is found, but the optimal value is not given explicitly to the CPLEX search procedure to avoid that the subsolver can infer information from the lower bound on the objective. If we reach a search node in which all variables in the set B are fixed but the sub-solver cannot conclude the infeasibility of the sub-problem or infer a integer feasible solution, we prune the search node and continue searching. If we explore the full partial tree on the set B without finding an optimal solution, we reject B. Again, there could have been an alternative search order in which succeeded with B. Our results are again lower bounds on the true probability of order-sensitive weak optimality backdoors.

If a set was rejected as an order-sensitive weak backdoor with this procedure, then we indeed explored the full partial tree over B. Hence, we know that for

[1] For practical reasons, we consider the dual-simplex algorithm as an efficient subsolver despite its exponential worst-case complexity; after all, the problem it solves lies in the complexity class P and dual-simplex is one of the most efficient practical procedures for this problem.

sure that B is not a traditional weak optimality backdoor. In addition, for every set that was accepted as order-sensitive weak backdoor, we record the values of the variables in B at the incumbent node with optimal value. We test whether assigning B to these values without prior search results in inferring an integer feasible solution of optimal quality. If yes, then the set is accepted as traditional weak optimality backdoor. If not it is rejected. Again, there can be false negatives due the fact that some other assignment different than the incumbent found could have sufficed. Therefore, our results on the probability that a set is a traditional weak optimality backdoor are lower bounds.

4.1 Smallest Backdoors

The size of the smallest traditional weak optimality and order-sensitive optimality-proof backdoor that we have found is presented in Table 1, representing an upper bound on the true smallest size. The values for the traditional weak optimality backdoor sizes are also an upper bound on the smallest order-sensitive weak optimality backdoor size. Note that the instance *10teams* does not have an optimality-proof backdoor because its objective value is already fixed in the problem specification. Overall, we find that the vast majority of instances have small or very small (traditional) weak optimality backdoors of less than 6% of the variables. For *air04* and *air05* we even find that setting less than one thousandth of the variables is already enough to enable the sub-solver to compute an overall optimal integer feasible solution! However, as the exceptions *pk1*, *pp08a* and *pp08aCUTS* show, some real-world MIPs might not exhibit small backdoors, even for very strong sub-solvers.

Table 1. Upper bounds on the smallest size of (traditional) weak optimality backdoors and of optimality-proof backdoors in absolute value and as percentage of the number of discrete variables in the problem instance

instance	variables	discrete variables	weak backdoors		orderOpt backdoors	
			size	%	size	%
10teams	2025	1800	10	0.56%	NA	NA
aflow30a	842	421	11	2.61%	85	20.19%
air04	8904	8904	3	0.03%	14	0.16%
air05	7195	7195	3	0.04%	29	0.40%
fiber	1298	1254	7	0.56%	5	0.40%
fixnet6	878	378	6	1.59%	5	1.32%
rout	556	315	8	2.54%	172	54.60%
set1ch	712	240	14	5.83%	28	11.67%
vmp2	378	168	11	6.55%	19	11.31%
pk1	86	55	20	36.36%	55	100.00%
pp08a	240	64	11	17.19%	47	73.44%
pp08aCUTS	240	64	11	17.19%	32	50.00%

4.2 Probability of Finding Small Backdoors

In addition to the smallest size of a backdoor, one is interested in knowing how hard it is to find small backdoor sets. One way to assess this difficulty is to estimate how many backdoor sets of a particular size exist for a given problem.

We want to approximate the probability that a set of variables of a given cardinality k is a backdoor. For each given backdoor size k, we sampled, uniformly at random, subsets of cardinality k from the discrete variables of the problem. For each set we evaluated whether it is a backdoor or not with the setup described in the beginning of this section.

We conducted this experiment many thousands of times for various cardinalities k. Figure 1 presents results for the instances *fiber* and *vpm2*. The curves labeled *orderOpt* refer to order-sensitive optimality-proof backdoors. The curves labeled *tradWeak* refer to weak optimality backdoors that are not order-sensitive. The curves labeled *orderWeak* refer to weak optimality backdoors that are order-sensitive. The curves labeled *tradWeak+orderOpt* refer to sets that are both (traditional) weak optimality backdoors and optimality-proof backdoors. Finally, the curves labeled *orderStrong* refer to sets that are both order-sensitive weak optimality backdoors and order-sensitive optimality-proof backdoors.

For the instance *fiber*, we observe that the probability that a set of a given size is an optimality-proof backdoor is much higher than the probability that a set of this size is a weak optimality backdoor. This evidence suggests that there are many more small optimality-proof backdoors than weak optimality backdoors. In addition, the probability that a set is both an optimality-proof backdoor and a weak backdoor is almost equal to the probability that it is a weak optimality backdoor. Our data shows that almost every set that was a weak optimality backdoor was also an optimality-proof backdoor. This suggests that for *fiber* the difficulty of the problem might lie in finding the optimal solution as opposed to proving its optimality.

Our study suggests that solving problems with a hardness profile similar to *fiber* can be significantly boosted by the availability of good initial solutions found by some heuristic search. This aligns well with the recent development of

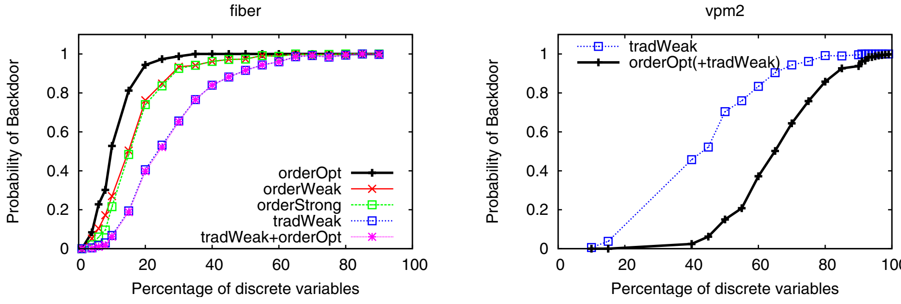

Fig. 1. Probability that a subset of variables of a given size is a backdoor when sampling uniformly

state-of-the-art MIP solvers for which it has been found that primal heuristics, so-called "feasibility pumps", can significantly boost performance.

For the instance *vpm2*, we have avoided displaying the order-sensitive weak backdoors because they fully overlap with the curve for the traditional weak backdoors. Contrary to *fiber*, for *vpm2* the probability that a set of a given size is a weak optimality backdoor is considerably higher than the probability that it is an optimality-proof backdoor. In addition, every set that was found to be an order-sensitive optimality-proof backdoor was also weak optimality backdoor. In other words, the curve for optimality-proof backdoors perfectly overlaps with the curve for sets that are both weak optimality and optimality-proof backdoors, including the curve for order-sensitive strong backdoors. We label the curve *orderOpt(+tradWeak)*. The results for *vpm2* give the intuition that the difficulty of the problem lies in proving optimality as opposed to finding an optimal solution.

To confirm the intuitions about the hardness profiles for solving *fiber* and *vpm2*, in Figure 2 we present the runtime distributions for *fiber* and *vpm2* in terms of the probability that a run is completed in a given number of search nodes. Three curves are presented for each instance. The curves labeled 'full run' represent the number of search nodes that it took to solve the problem fully - both find an optimal solution and prove its optimality. The curves 'opt soln run' represent the number of search nodes that were explored before the incumbent solution had the known optimal value. The curves 'proof run' capture the number of search nodes that were explored to prove that a solution of a better quality does not exist, i.e., proving infeasibility once an optimal solution is provided. This comparison allows us to estimate the relative effort spent on each task and the effort overall. We see that the intuition from the distribution of backdoor sizes was indeed correct. For *fiber*, the effort spent of finding the optimal solution explains almost all of the full runtime, while the effort that is needed when only proving infeasibility is considerably less. On the other hand, for *vpm2* the gap between the effort on finding an optimal solution and the full effort is substantial, especially in the beginning. The full runs are clearly taking much longer than the

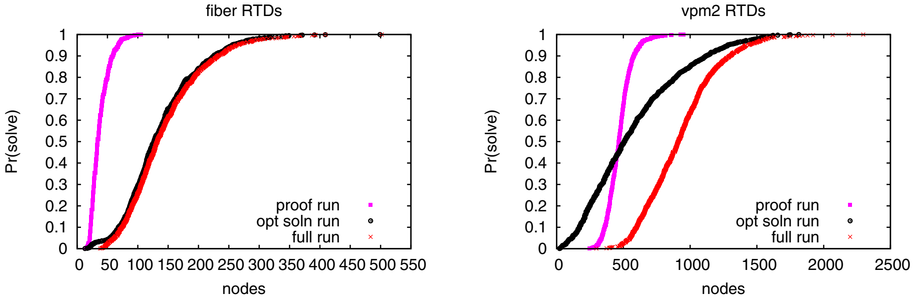

Fig. 2. Runtime distributions for finding an optimal solution, for proving optimality, and for fully solving the problem

fastest solution-finding runs, but about the same as the slowest solution-finding runs. Here, proving optimality takes longer than the fastest solution-finding runs but shorter than the slowest solution-finding runs.

4.3 LP Relaxations as Primal Heuristics

We saw that MIPs, even when they have small backdoors, may only have very few weak backdoor sets of a particular (small) size. The question arises of how a MIP solver could exploit its sub-solver to find small backdoors. To see whether LP relaxations can provide guidance about which variables may belong to a small backdoor set, we slightly modified the experiment from the previous section. Rather than sampling sets of desired cardinality by selecting variables uniformly at random, we biased the selection based on the "fractionality" of variables in the root relaxation. The fractionality of a variable measures how far it is from the nearest integer value. E.g., the fractionality of a variable X with domain 0,1 is simply $f(X) = min\{|x|, |1-x|\}$. More formally, if the root LP value of variable X_i is $\bar{x_i}$, then its fractional part is $f_i = \bar{x_i} - \lfloor \bar{x_i} \rfloor$. We assign to each variable a weight $f(X_i) \leftarrow 0.5 - |0.5 - f_i|$. Note that the quantity $f(X_i)$ captures the "infeasibility" of a variable which is a well-known measure for picking branching variables in mixed integer programming. Some discrete variables could be integral in the root LP. For such variables X_i, we assign a small non-zero weight $f(X_i) = \varepsilon$. After we normalize the variable weights, we choose a subset of size k where each variable is selected with probability proportional to its normalized weight.

For each desired size k, we sampled many sets of variables again and tested which ones were backdoors. The result of these experiments is summarized in Figure 3 for *fiber* and in Figure 4 for *vpm2*. The effect of sampling sets in a biased fashion is clearly visible (curves resulting from biased selection are marked with a "b-"). For the instance *fiber*, choosing sets biased by the root LP clearly increases the probability of selection a set which is an optimality-proof backdoor or a set which is a weak optimality backdoor. Surprisingly, selecting 6% of the variables

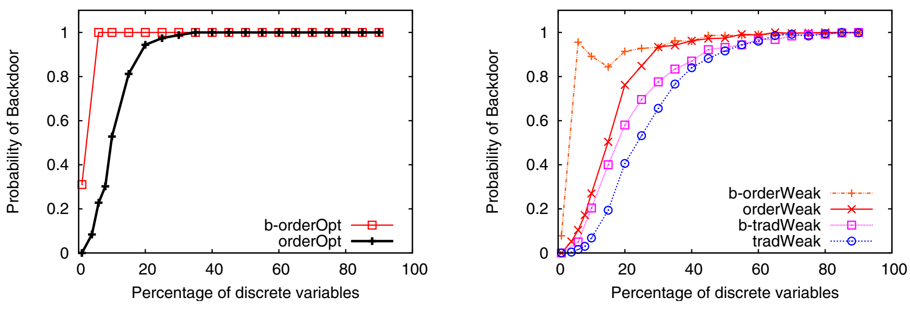

Fig. 3. FIBER: Comparing the probability that a subset of variables of a given size is a backdoor when sampling uniformly versus when sampling based on the fractionality of variables at the root

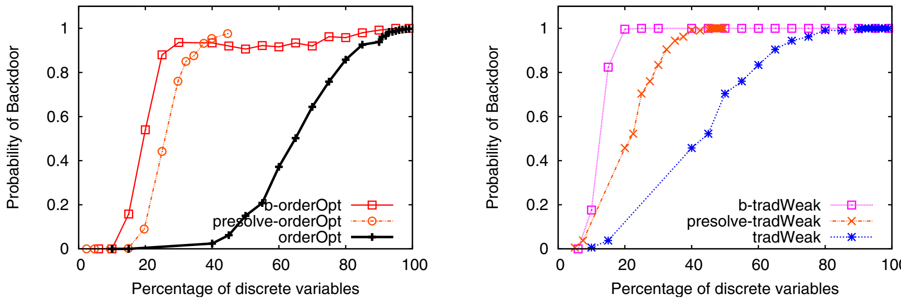

Fig. 4. VPM2: Comparing the probability that a subset of variables of a given size is a backdoor when sampling uniformly versus when sampling based on the fractionality of variables at the root

in this fashion is enough to guarantee that the set is an optimality-proof backdoor (100%), and give a 95% chance that the selected set is a weak backdoor.

The improvement effect is even more dramatic for the instance *vpm2*. Here, with 20% of the variables selected in the biased way we are guaranteed to select a weak backdoor, compared to a less than 2% chance when selected uniformly. Also, while with 30% of the variables selected in the biased way we have a 93% chance of selecting an optimality-proof backdoor set, we have less than 0.02% chance of such event when selecting uniformly. This shows clearly that an LP sub-solver can be exploited effectively to find small backdoors.

One thing to note is that before solving the root LP, CPLEX applies a *pre-processing* procedure which simplifies the problem and removes some variables whose values can be trivially inferred or can be expressed as an aggregation of other variables[2]. This procedure can sometimes result in dramatic reduction in the effective problem size. In *fiber*, the discrete variables removed by preprocessing are less than 17%. However, for *vpm2* the preprocessing removes 50% of the discrete variables.

One advantage of biasing the set selection by the root LP is that the variables trivially inferred by the preprocessing will have integral values, and will be selected only with some very small probability. To evaluate whether the biased selection draws its advantage over the uniform selection solely on avoiding pre-processed variables, we evaluated the probability of selecting a backdoor set when sampling uniformly among only the discrete variables remaining after pre-processing for *vpm2*. The results for this experiment are presented in the curves *presolve-orderOpt* and *presolve-tradWeak* in Figure 4. These curves show that choosing uniformly among the remaining variables is more effective for finding backdoors than choosing uniformly among all discrete variables, but it is not as good as the biased selection based on the root LP relaxation. Hence biasing the selection by the fractionality of the variables of the root LP has additional merit for discovering small backdoor sets.

[2] However, the user-defined branching procedure of CPLEX still works on the original set of variables.

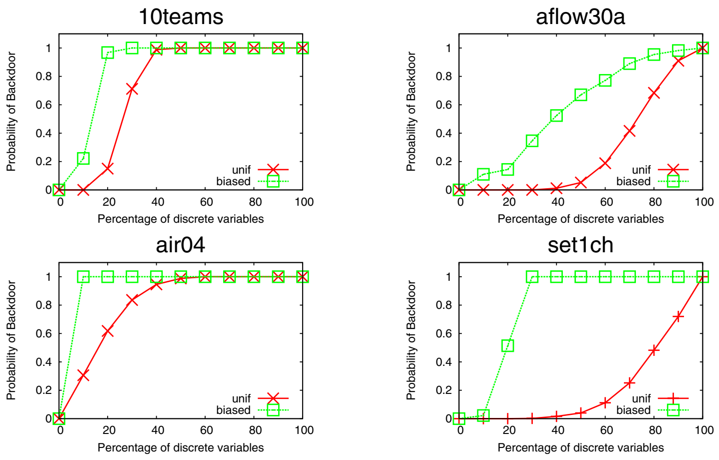

Fig. 5. Probability that a subset of variables of a given size is a traditional weak optimality backdoor backdoor when sampling uniformly (crosses) versus when sampling based on the fractionality of variables at the root (biased)

Other MIPLIB instances for which we have found that the biased selection has a substantial effect are *10teams*, *aflow30a*, *air04*, and *set1ch*. We present the results in Figure 5. For these instances, we only performed a quick evaluation, where we tested whether a set of variables B is a traditional weak optimality backdoor by setting their values to the values in the optimal solution found by default by CPLEX. Hence, the results are loose lower bounds on the actual probabilities.

5 Conclusion

In this work, we extended the concept of backdoor sets from constraint satisfaction problems to combinatorial optimization problems. This extension also involved incorporating learning into the notion of backdoors by introducing order-sensitive backdoors. While it has been previously shown that real-world SAT instances have very small backdoors, here we showed that small backdoors also exists to standard benchmark instances in mixed integer programming. In particular, optimization instances can have very small weak optimality backdoors and often also small optimality-proof backdoors. Surprisingly, sometimes the optimization-proof backdoors can in fact be smaller than the weak optimality backdoors.

We also considered the question of how hard it is to find small backdoor sets and provided extensive numerical results. We studied the probability that a set of a given size is an order-sensitive optimality-proof backdoor and the probability that it is an order-sensitive or traditional weak optimality backdoor. In general, we have shown that the difference in the distributions of weak optimality backdoors and of optimality-proof backdoors for a particular instance is well aligned with the difference in the runtime distributions for the tasks of finding an optimal solution and proving optimality, respectively. Finally, we have also demonstrated that the fractionality of variables in the root LP relaxation is a very good heuristic for uncovering small backdoors for both solution finding and for proof of optimality.

Acknowledgments

This research was supported by IISI, Cornell University (AFOSR grant FA9550-04-1-0151), NSF Expeditions in Computing award for Computational Sustainability (Grant 0832782), NSF IIS award (Grant 0514429), and the Cornflower Project (NSF Career award 0644113). The third author was partly supported by NSERC Postgraduate Fellowship. Part of this work was done while the forth author was visiting McGill University.

References

[1] Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Operations Research Letters 34(4), 1–12 (2006), `http://miplib.zib.de`

[2] Bixby, R.E.: Solving real-world linear programs: A decade and more of progress. Oper. Res. 50(1), 3–15 (2002)

[3] Dilkina, B., Gomes, C.P., Sabharwal, A.: Tradeoffs in the complexity of backdoor detection. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 256–270. Springer, Heidelberg (2007)

[4] Gent, I.P., Walsh, T.: Easy problems are sometimes hard. AI J. 70, 335–345 (1994)

[5] Gomes, C.P., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 121–135. Springer, Heidelberg (1997)

[6] Gomes, C.P., Selman, B., McAloon, K., Tretkoff, C.: Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In: 4th Int. Conf. Art. Intel. Planning Syst. (1998)

[7] Hogg, T., Williams, C.: Expected gains from parallelizing constraint solving for hard problems. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994), Seattle, WA, pp. 1310–1315. AAAI Press, Menlo Park (1994)

[8] ILOG, SA. CPLEX 10.1 Reference Manual (2006)

[9] Li, C.M., Anbulagan: Heuristics based on unit propagation for satisfiability problems. In: 15th IJCAI, Nagoya, Japan, pp. 366–371 (August 1997)

[10] Smith, B.M., Grant, S.A.: Sparse constraint graphs and exceptionally hard problems. In: 14th IJCAI, Montreal, Canada, vol. 1, pp. 646–654 (August 1995)

[11] Williams, R., Gomes, C., Selman, B.: Backdoors to typical case complexity. In: 18th IJCAI, Acapulco, Mexico, pp. 1173–1178 (August 2003)

[12] Williams, R., Gomes, C., Selman, B.: On the connections between heavy-tails, backdoors, and restarts in combinatorial search. In: 6th SAT, Santa Margherita Ligure, Italy, pp. 222–230 (May 2003)