Clairvoyant Restarts in Branch-and-Bound Search Using Online Tree-Size Estimation

Daniel Anderson,[1][*] **Gregor Hendel,**[2] **Pierre Le Bodic,**[3] **Merlin Viernickel**[2]

[1]Department of Computer Science, Carnegie Mellon University, Pittsburgh, USA
[2]Mathematical Optimization Department, Zuse Institute Berlin, Berlin, Germany
[3]Faculty of Information Technology, Monash University, Melbourne, Australia
dlanders@cs.cmu.edu, hendel@zib.de, pierre.lebodic@monash.edu, viernickel@zib.de

Abstract

We propose a simple and general online method to measure the search progress within the Branch-and-Bound algorithm, from which we estimate the size of the remaining search tree. We then show how this information can help solvers algorithmically at runtime by designing a restart strategy for Mixed-Integer Programming (MIP) solvers that decides whether to restart the search based on the current estimate of the number of remaining nodes in the tree. We refer to this type of algorithm as *clairvoyant*. Our clairvoyant restart strategy outperforms a state-of-the-art solver on a large set of publicly available MIP benchmark instances. It is implemented in the MIP solver SCIP and will be available in future releases.

Introduction

A standard technique in backtracking search consists in restarting the search at the root node, retaining as much information about the previous tree as efficiently possible. This is common practice in Constraint Programming (CP) and Satisfiability (SAT) solvers, for instance. In contrast, restarts are rarely used in modern Mixed-Integer Programming (MIP) solvers. Instead, the entire search is usually performed within a single Branch-and-Bound (B&B) tree without restarting. In the MIP solver SCIP, for instance, restarts may only be performed at the root node.

As far as we are aware of, restarts in MIP have only been studied by (Achterberg 2007b), using as a criterion the number of globally fixed variables. This criterion therefore ensures that presolvers would fix some variables after a restart, which would have a positive cascading effect on the rest of the search. Hence, restarts were then classified as a type of presolving technique. However, using this criterion, restarts appeared to be detrimental to performance. The author concluded that "in order to make good use of delayed restarts, one has to invent different criteria for their application".

However, besides presolving, restarts may also be beneficial to branching strategies. Indeed, the branching decisions taken at the start of the B&B search have a significant impact on the efficiency of the entire search. Unfortunately, without restarts, these decisions are quite uninformed, as little search has been performed yet. Better decisions at the top of the tree

can generally be made after a restart, as more information is available then.

Thus we started this study with a simple experiment: in SCIP, we forced a restart after 1000 nodes. On the MMMc benchmark, defined below, this strategy yielded an average 4.7% slow down. However, by only selecting instances for which default SCIP requires at least 50k (resp. 100k, 200k, 500k) nodes, then this forced restart produces a speed-up of 7% (resp. 9, 17, 18%) on the 106 (resp. 88, 72, 48) remaining instances. Indeed, for instances that require relatively small trees, poor decisions at the top of the tree have a smaller impact, and thus restarts are generally not beneficial, but this trade-off becomes interesting for large instances. Hence it appears that if the final tree size of the current B&B tree could be known during the search, it would be a good criterion for restarts. In this paper we will show how estimates of the tree size can be obtained and used as a criterion for deciding restarts, improving the overall performance of the MIP solver SCIP.

Related Work and Discussion

Knuth was the first to attempt to measure the time taken by backtracking searches (Knuth 1975). Knuth's model assigns probabilities to each potential branch of the backtracking procedure that estimate the relative sizes of the child subtrees in each branch. The method then computes an estimate of the total tree size by sampling root-to-leaf paths subject to these probabilities. Knuth showed that even when naively using uniform probabilities, this sampling procedure produced reasonable results, but the high variance of the method makes it unreliable, especially when the tree is unbalanced (Kilby et al. 2006). Many improvements have been proposed, notably in (Purdom 1978), (Chen 1992) and (Lelis, Otten, and Dechter 2013).

Kullman extended Knuth's ideas by augmenting the estimation with additional information about the progress of the search (Kullmann 2009). Provided that one has a measure of how much progress is made by a particular decision, a quantity called the τ-value can be derived that can be used to estimate the relative sizes of the child subtrees of a particular search node. Kullman studied this problem in the context of SAT, where for example, the number of satisfied clauses can be used as a sufficient progress measure. Kullman showed that this method could be used to derive probabilities that

---

reduce the variance exhibited by Knuth's sampling method.

A similar model was studied for MIP, where a suitable progress measure on the search tree is the dual gap change as branching decisions are made (Le Bodic and Nemhauser 2017). They derived a quantity that describes the asymptotic tree sizes, referred to as the ratio φ, which is similar to Kullman's τ-value. In (Belov et al. 2017), this model is applied to tree-size estimation for MIP, where it is shown that the use of the φ value for deriving subtree weights roughly halves the error in the estimate in the sampled tree size compared to using Knuth's uniform probabilities.

The methods we have reviewed so far tackle the problem of *offline* sampling. While these methods can to some degree be extended to the online case (see e.g. (Belov et al. 2017)), some information is lost in the process. For instance, one way to adapt offline sampling to online tree-size prediction is to treat the leaves obtained by the tree search procedure as if they had been obtained randomly. However, while in offline sampling, samples are drawn independently, this does not hold when obtaining leaves online. This phenomenon clearly materializes in the difference between offline and online results of (Belov et al. 2017). Indeed, at any given point in the search, supposing that samples are independent equates to supposing that the first or latest samples observed are equally good predictors of the next samples to be observed. In other words, any such method would ignore possible *trends* in the series of samples. However, we argue that there are multiple types of trends affecting the samples obtained in the B&B. First, since the depth of the tree grows as the search progresses, increasingly deeper leaves are found, although this is not a monotonic process. Second, and conversely, after a primal solution is found which improves the primal bound, nodes can be pruned at shallower depths than previously. A similar phenomenon occurs with strong conflicts (Achterberg 2007a). Other factors such as "smart" node selection strategies (Berthold, Hendel, and Koch 2017) contribute to creating varying trends in the amount of resources required to reach a leaf. Hence, in an online setting, while we cannot suppose that samples are independent, capturing trends may mitigate this loss.

Two online methods for tree-size estimation are given in (Kilby et al. 2006), but as we will see, they do not detect or exploit trends. Given the multiset D of the depths of the leaves of a binary tree traversed at a given point in the search, the *Weighted Backtrack Estimator* (WBE) returns $\frac{2|D|}{\sum_{d \in D} 2^{-d}} - 1$. The denominator equals 1 if D is the multiset of depths of the entire set of leaves of the tree, hence the WBE is unbiased. The authors consider the case where all leaves have depth $d \gg 1$, except the left child of the root node, which is a leaf. Suppose the leaf with depth 1 is visited first. After two samples, one with depth 1, the other d, WBE computes a tree-size estimate of $\frac{4}{2^{-1}+2^{-d}} - 1 \approx 7$, which can be arbitrarily far from the $d+1$ nodes already traversed to reach the sample of depth d. Until the number of samples approaches 2^{d-1}, the sample of depth 1 will render WBE essentially useless: for a large enough depth d, the estimate is approximately $4|D| - 1$. As pointed out in (Le Bodic and Nemhauser 2017), the 0.5 weight of the left child encodes

an initial implicit assumption that both sides of the tree have the same size. After finding a sample at depth d on the right side of the tree, it should become clear that the assumption was incorrect. However, this is not taken into account by the WBE other than by the slow incorporation of other samples with virtually insignificant individual weight. While this example is clearly pathological, unbalanced trees are the standard in MIP, CP, and many other fields. This pathological instance will be a recurring feature throughout the paper.

The authors also introduce the *recursive estimator* (RE), which at any inner node does the following: if both its children have unkown size, the node has unkown size. If the size of a single child is unknown, estimate it to be the same as the other child's. If both children have a known or estimated size, add them. Leaves have known size 1. As soon as one leaf is found, an estimate propagates upwards recursively to the root of all subtrees it belongs to, and thus to the root node. The RE recovers from the pathological example given for the WBE after 1 sample of depth d. Indeed, it abandons the hypothesis that a 0.5 portion of the subtree lies on each side of an inner node as soon as at least one sample is available on each side. It is clear why RE performs better than WBE on very unbalanced trees, as shown in (Kilby et al. 2006). However, if at a node of a reasonably balanced tree, the left child is fully explored, and the right child has a single sample, both sizes are simply added, giving the same "weight" to the right sample as to all left samples taken together, which creates higher variability.

Both WBE and RE suffer from an additional limitation in the case of optimization problems. When an improving solution is found, new samples will reflect it, but no mechanism revisits the estimates provided by previous samples, despite the fact that in practice, many nodes are pruned by bound, hence they would have been pruned at shallower depth, yielding smaller estimates. This can be fixed if the entire tree is kept in memory and reprocessed to compute new estimates upon improvement of the primal bound. However, this is more memory than the solving process itself keeps, which only requires the open parts of the tree. In our tests, this created a significant time and memory overhead in the search itself.

Online methods specific to the B&B include a statistical model of the shape of the B&B tree (Cornuéjols, Karamanov, and Li 2006), and (Özaltın, Hunsaker, and Schaefer 2011), which defines a progress measure, the sum of subtree gaps, and uses double exponential smoothing, a time series forecasting technique, to obtain a tree-size estimate.

Note that tree-size estimates have been extensively studied for A^*, see e.g. (Thayer, Stern, and Lelis 2012) and references therein. In particular, this reference reviews concepts of progress measures and velocity-based estimates for A^*.

Contributions

Our first contribution is a formal definition of an online progress measure, which does not require a significant time or memory overhead, and is invariant to changes in primal bound. Second, we formally define a new acceleration-based tree-size estimation method, which we show generalizes and extends multiple existing tree-size estimation meth-

ods (WBE, velocity-based methods). The combination of progress measure and tree-size estimates is loosely coupled, allowing new variants to be readily defined.

Our main contribution is the integration of tree-size estimates into a *clairvoyant* restart strategy. Note that this is different from algorithm selection (Lobjois and Lemaître 1998; Kilby et al. 2006), wherein online estimates are used to select an algorithm before the actual search, and always incurs a fixed overhead. The clairvoyant restart strategy is truly online: it observes the default search and *may* decide to restart. If it does not, there is no measurable memory or time overhead. If it does, our benchmarks shows that it improves the run time by 8%, on average. Over all instances, the average time improvement is 4%, and 10% over "hard" instances, which is significant for MIP. These results, together with the simplicity of the method, demonstrates the general potential of clairvoyant algorithms.

Tree Search Progress Measures

A Simple and General Progress Measure

Throughout the paper we will suppose that the input is an online sequence of $m \geq 2$ rooted trees T_k, which represent the m states of the final B&B tree $T = (V, A)$ from the beginning of the search (T_1) to its end ($T_m = T$). We will refer to $T_1, \ldots, T_{m-1}$ as *intermediate* trees, and to T_m as the *complete* tree. All trees are rooted at the same root r, and for all $k \in \{1, \ldots, m\}, T_k$ is a subgraph of T_{k+1}. For $k \in \{1, \ldots, m\}$, we denote as:

- $V_k \subseteq V$ the set of nodes of T_k, i.e. the nodes of T which have been explored at step k,

- $I_k \subset V_k$ the set of inner nodes of T_k (which includes the root r if r is not a leaf),

- $L_k \subseteq V_k$ the set of leaves of T_k,

- $F_k \subseteq L_k$ the set of *final* leaves of T_k, i.e. base cases of the search.

Between two successive trees T_k and T_{k+1} at some $k \in \{1, \ldots, m-1\}$, we suppose that the following operations (or combination thereof) are possible:

- Solve a non-final leaf: $i \in L_k \setminus F_k$ and $i \in F_{k+1}$, i.e. a non-final leaf is proven to be final.

- Expand/branch on a non-final leaf: $i \in L_k \setminus F_k$ and $i \in I_{k+1}$, i.e. a non-final leaf becomes an inner node.

Proposition 1. *For all $k \in \{1, \ldots, m-1\}$ (corresponding to the indices of intermediate trees),*

1. *$I_k \subseteq I_{k+1}$, i.e. the set of inner nodes is non-decreasing,*
2. *$F_k \subseteq F_{k+1}$, i.e. the set of final leaves is non-decreasing,*
3. *$F_k \subsetneq L_k$, i.e. there exists a non-final leaf in any intermediate tree,*
4. *$F_k \subset F_m = L_m$, i.e. all leaves are final only at T_m.*

For simplicity, we will use the sets V, I, L, F without subscript to refer to nodes of the complete tree T_m, which includes all nodes from $T_k, k \in \{1, \ldots, m-1\}$.

In order to compute a progress measure for a tree T_k, the methods we study require a measure of the *hardness* h_i to solve a node $i \in V$, which we define as follows:

(a) Hardness scheme p_u.

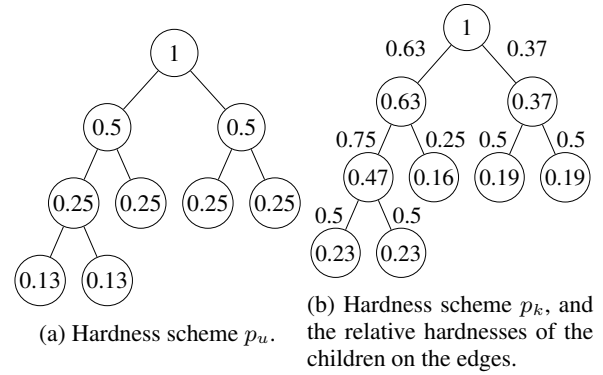(b) Hardness scheme p_k, and the relative hardnesses of the children on the edges.

Figure 1: Two search trees depicting the hardness values of its nodes with different hardness schemes. Numbers are rounded to two decimal places.

- The hardness value at the root node r is $h(r) = 1$,
- The hardness value $h(i)$ of an inner node $i \in I$ is equal to the sum of the hardness values of its children:

$$h(i) = \sum_{ij \in A} h(j),$$

where $h(j) > 0$ for all $ij \in A$.

The recursive definition of hardness can be seen as a simple partition, allocation, or repartition of the hardness of a subproblem into the subproblems it is divided into. In the example given in Figure 1a, the hardness value at a given node is uniformly divided among its children, but this is not necessary. In fact, a hardness *scheme* can be defined similarly to a probability scheme, as in (Belov et al. 2017), by assigning for every $ij \in A, h(j) = p(ij) \cdot h(i)$, where p can be uniform (then denoted as p_u), or approximate the tree balance (p_k), as shown in Figure 1b. Note in the case where the hardness scheme is uniform, the notion of hardness of a leaf corresponds exactly to its weight in Knuth's offline tree sampling.

Given a set of nodes $X \subseteq V$, we define its hardness as

$$h(X) = \sum_{i \in X} h(i).$$

There are three direct results which make this notion of hardness useful for the purpose of measuring the search progress.

Proposition 2. *For any two trees T_k and T_l, with $k, l \in \{1, \ldots, m\}, k \neq l$, we have $h(L_k) = h(L_l)$.*

Proof. From the definition of the hardness of nodes, $h(L)$ remains constant when adding children under a leaf of T, thus the property holds by induction. $\square$

In other words, the hardness at the leaves remains constant throughout the search.

Corollary 1. *For any $k \in \{1, \ldots m\}$, $h(L_k) = 1$.*

Proof. By definition, $h(r) = 1$, and $L_1 = \{r\}$, hence the property is true for T_1, and thus holds for all k's via Proposition 2. $\square$

We simply define the progress measure of a search tree T_k as $h(F_k)$, the sum of the hardness at the final leaves. With uniform allocation of hardness to children, $h(F_k)$ corresponds to the denominator of the WBE of (Kilby et al. 2006).

Proposition 3. *The following (in)equalities hold:*

$$0 = h(F_1) \leq \cdots \leq h(F_{m-1}) < h(F_m) = 1$$

Proof. Since we suppose $m \geq 2$, the root r is not a final leaf in T_1, hence $F_1 = \emptyset$. Further, for any $k \in \{1, \ldots, m-1\}$, $F_k \subseteq F_{k+1}$, hence $h(F_k) \leq h(F_{k+1})$. Finally, for any $k \in \{1, \ldots, m-1\}$, $F_k \subset L_k$, and $\forall i \in V, h(i) > 0$, hence $h(F_k) < h(L_k) = 1$, and $F_m = L_m$, thus $h(F_m) = h(L_m) = 1$. $\square$

The hardness of the set of final leaves hence provides a non-decreasing measure of the progress of a Backtracking search.

Ideally, a progress measure would be linear from 0 to 1, in which case a linear extrapolation provides a perfect estimate of m at any given $k \geq 2$.

Computational Assessment of Progress Measures

We now perform computational experiments on progress measures for B&B on Mixed-Integer Programming (MIP) instances. Throughout the paper we use the MIP solver SCIP 6.0 (Gleixner et al. 2018) and the so-called MMMc MIP benchmark, a collection of 496 instances from MIPLIB 2010 "benchmark" (Koch et al. 2011), MIPLIB 2003 (Achterberg, Koch, and Martin 2006), MIPLIB 3.0 (Bixby et al. 1998), and COR@L (Coral 2010). We solved each instance in MMMc with default SCIP and output the B&B tree in the *vbc* format, as in (Belov et al. 2017). We then discard the trees corresponding to instances that were not solved within the time limit, and converted the vbc files to a more compact format, leaving 349 instances. We then simulate the traversal of the B& B tree, and compare the value of the progress measure to the true fraction of nodes that have been explored.

We can see in Figure 2 that both progress measures can be observed to be reasonably accurate, and produce very similar results. *Actual* represents the true progress of the search in terms of the number of nodes visited thus far, while p_k and p_u correspond to the hardness schemes previously mentioned.

We now provide some quantitative assessments. We measure the average error exhibited by the approximations as the average difference between the actual progress and the predicted progress. The results are shown in Table 1. For both the uniform and domain-based probabilities, the average errors tend to be about 15%. On large instances, which are the primary instances of interest to us, the errors are slightly larger, reaching around 20%, which is still very good.

Although the estimation based on p_k is more accurate than p_u, it has several implementation shortcomings that need to be considered. Note that for our simulation, we use the exact LP objective value of a node, which can only be known after the node LP relaxation has been solved. For p_k, solving a node LP relaxation also affects the subtree of that node's sibling, potentially including previously solved leaves within
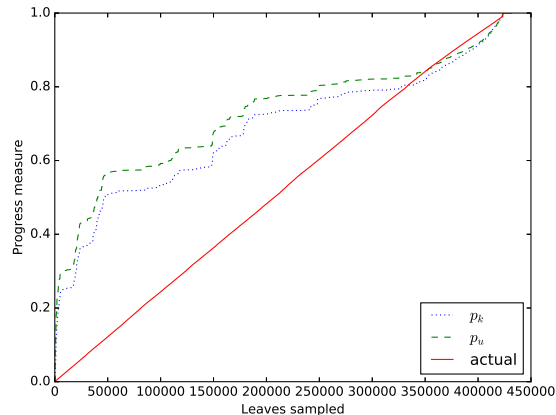
(a) Instance NOSWOT.
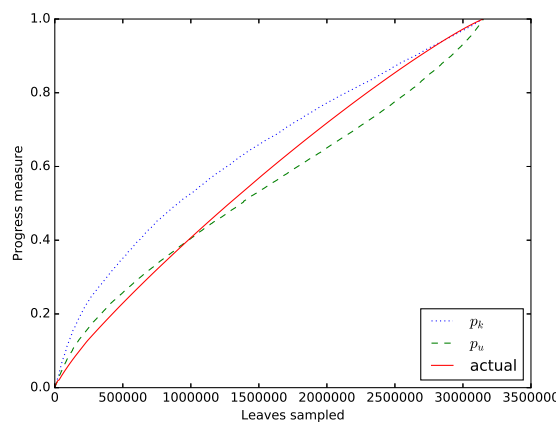
(b) Instance MAS74.

Figure 2: Progress measures for two MIP instances.

that subtree. Revisiting already solved subtrees is not only time consuming, but it has the additional disadvantage that the current progress may decrease. Therefore, the hardness of a node must be determined at creation time, where only approximations of the expected lower bounds are available, either as a result of strong branching, or based on the pseudo cost information of the branching variable. For performance considerations, strong branching is only applied at the beginning of the search (and hence at the topmost nodes of the tree). Thus, the majority of the hardness measure must be based on potentially very inaccurate pseudo-cost information. Given these drawbacks, and the fact that the p_k scheme yields only marginally better simulations under ideal conditions, we decided to proceed with uniform probabilities p_u for the remainder of our experiments.

From Search Progress to Tree-Size Estimate

A Linear Extrapolation Technique

We suppose that we possess a measure r_k of the resources required from the beginning of the solve until T_k is reached.

	All instances		Large (≥ 1000 nodes)	
	Average	Median	Average	Median
p_u	0.17	0.14	0.24	0.24
p_k	0.16	0.12	0.20	0.17

Table 1: The average and median of the average errors exhibited by the progress measures using domain-based probabilities (p_k) and uniform probabilities (p_u).

In our experiments, we have chosen r_k to be the number of nodes explored up to and including step k, but it can be defined to be time, for instance, or any other relevant measure. Throughout the rest of the paper we will use the shorthand h_k for the progress measure $h(F_k)$ for any tree $T_k, k \in \{1, \ldots, m\}$.

Perhaps the most natural estimate of the remaining amount of resources required to reach T_m while at T_k would be to suppose that each % of progress requires a fixed amount of resources. This leads to the formula

$$\hat{r}(k) = r_1 + \frac{r_k - r_1}{h_k}, \tag{1}$$

where $\hat{r}(k)$ is the estimation at step k of the total amount of resources r_m required to obtain the complete tree T_m. Equation (1) provides an exact estimate if the assumption that the amount of resources required for each unit of progress is fixed.

Unfortunately, in practice, a linear progress measure cannot be obtained. Indeed, for a feasibility problem, the search can stop at any point if a feasible solution is found. A similar phenomenon is at play for an optimization problem if a solution is found whose objective value matches the dual bound.

The estimate given by (1) is equivalent (up to a constant) to the WBE if $r_1 = 1$, $r_k = 2|F_k| + 1$ and a uniform hardness scheme. If we choose instead $r_k = |V_k|$, the pathological example for WBE returns an estimate of size $4 + 2d$ instead of 7 after the same two samples. This demonstrates that a given progress measure can lead to tree-size estimates of varying quality, and that it may be beneficial to decouple these concepts in order to study them.

A Velocity-Based Technique

We now study how to handle a non-linear progress measure, i.e. the fact that one unit of progress may require nonconstant resources throughout the search. We find that it helps to express the problem of estimating the amount of resources required to complete the search using a simple physics analogy, considering an object moving from point a to b, at distance 1 from each other. The object departs a at time step $k = 1$ and reaches b at $k = m$. Suppose that the velocity of the object is constant, then at time k the average velocity and the instantaneous velocity of the object are equal at any point between a and b. Using our notation, at

step k, this translates to

$$\frac{h_k - h_1}{r_k - r_1} = \frac{h_k - h_{k-1}}{r_k - r_{k-1}}.$$

Estimating the time of arrival of the object using either of these velocities is therefore equivalent. Equation (1) is simply the formula of total distance ($h_m = 1$) over average velocity, plus initial displacement.

If the velocity is not constant, then it is not clear whether the average velocity, the instantaneous velocity, or perhaps another measure would yield the most accurate estimate. Inspired by time-series forecasting, we therefore propose to compute a simple moving average technique, which can be parametrized to compute the velocity on an arbitrary number of most recent observations. At step k, given a window size $0 < s < k$, it computes the average velocity over the last $s + 1$ observations:

$$v_{k_0,k} = \frac{h_k - h_{k_0}}{r_k - r_{k_0}},$$

where $k_0 = k - s$ is the start of the window. The total amount of resources can then be estimated by the displacement formula

$$\hat{r}(k) = r_k + \frac{h_m - h_k}{v_{k_0,k}}. \tag{2}$$

In our recurrent pathological example, with $r_k = 2|F| + 1$, the estimate (2) returns the exact tree size $2^d + 1$ at any point of the search as long as the window excludes the left child. This is because the velocity of the search is constant when traversing the leaves at depth d.

An Acceleration-Based Technique

Taking the physics analogy further, we propose to capture the acceleration (positive or negative) of the progress measure, in order to estimate the change in velocity in future observations. In principle, this would allow us to capture the fact that as the leaves are found at gradually increasing depths, the velocity of the search decreases.

Let k_0, k_1, k_2 be the start, middle and end of the window of size s, respectively. The acceleration in the window can then be computed as

$$a_k = 2 \frac{v_{k_0,k_2} - v_{k_0,k_1}}{r_{k_2} - r_{k_1}}.$$

The velocity v over the entire window is corrected by the acceleration a_k using the formula

$$v = v_{k_0,k_1} - 0.5 a_k (r_{k_0} + r_{k_1}).$$

Supposing that the acceleration remains at its value a_k after step k, the estimation of the total amount of resources $\hat{r}(k)$ can be obtained by solving the displacement formula under constant acceleration

$$1 = h_m = h_k + v \cdot (\hat{r}(k) - r_k) + a_k (\hat{r}(k) - r_k)^2 / 2,$$

which is a simple quadratic equation for which every data but $\hat{r}(k)$ is known at step k. Its solution is

$$\hat{r}(k) = r_k + \max \left(\frac{-v \pm \sqrt{v^2 + 2 a_k (1 - h_k)}}{a_k} \right),$$

where $\max$ returns the only positive root of this quadratic equation.

	All instances	Large (≥ 1000 nodes)
No Window		
No acceleration	16%	43%
Acceleration	21%	49%
Window		
No acceleration	14%	187%
Acceleration	13%	48%

Table 2: The median of the average percentage errors exhibited by the tree-size estimates with and without the acceleration technique.

Computational Assessment of Tree-Size Estimates

Simulations on Pre-Generated Trees   We provide quantitative assessments similar to those for the progress measure in order to assess whether the estimates predicted by the velocity technique are useful. We compare the use of a window size 100, and no window at all, and measure the average error between the predicted tree size and actual tree size as a percentage of the actual tree size. We do not measure the error for the first 100 leaves, since the variance of the estimate makes it unreliable before sufficiently many samples are taken. We also perform a sliding window minimum over the predictions in order to mitigate the fact that occasionally, a massive over prediction will be computed by a single sample, only to be corrected a few samples later. This emulates the fact that in our later MIP experiments, we will not perform a restart unless high estimates are produced for many samples in a row. We use a sliding window minimum with a window size of 50.

For the velocity-based technique and the acceleration-based technique, we show the median percentage error of the tree-size estimates in Table 2. We do not report average percentage errors since a few outlying instances skew the results and make the quantity useless. Table 2 shows that the errors tend to be at most 50% for large tree instances, which means that we are correctly predicting the order of magnitude of the tree sizes, which should be accurate enough to inform a restart strategy. The presented results confirm that the acceleration method is clearly beneficial within a moving window. The exceptionally poor performance of the window method without acceleration requires further investigation.

A Clairvoyant Restart Strategy

Current restart strategies in MIP solvers

To the best of our knowledge, MIP solvers such as CPLEX, Gurobi and SCIP currently only restart the solution process at the root node. (From its logging message "Resetting tree to root", it appears that the solver FICO Xpress may apply restarts during the search, but this is all we know.) At the root node, information obtained from the initial solution to the LP relaxation, valid cutting planes, or improving solutions may lead to many variable domain reductions that were not detected at presolving time. If the percentage of fixed

integer variables exceeds a threshold (by default, 2.5 % in SCIP), this justifies a restart of the solution process. During a restart, SCIP preserves variable domain reductions, solutions, valid cuts and branching history information (Achterberg and Berthold 2009). By default, SCIP may perform arbitrarily many root node restarts.

As we have reported before, (Achterberg 2007b) conducted MIP experiments where restarts were allowed during the B&B tree search, where restarts would be decided based on the number of variables which have been fixed globally. The results, however, were inferior compared to the SCIP default strategy. As (Achterberg 2007b) points out, additional global variable fixings seem to occur mostly at the final stage of the solution process, when the computational overhead of rebuilding a new search tree from scratch becomes too high. Our clairvoyant restart strategy addresses this disadvantage by using tree-size estimation instead of the number of global variable fixings.

A Clairvoyant Restart Strategy

For the remainder of this work, we use $r_k = |V_k|$ as the resource measure. We implement our clairvoyant restart strategy as follows: given a parameter $\gamma > 1$, a restart is decided at step k if

$$\gamma \cdot r_k < \hat{r}(k). \tag{3}$$

In our experiments, γ is set to 100, and we update the estimate $\hat{r}(k)$ at every leaf node, i.e. at every step k such that $|F_k| = |F_{k-1}|+1$.

We have two safeguards against the potentially high variance of $\hat{r}(k)$. First, a restart is only triggered after condition (3) is satisfied for 50 consecutive k's. Second, no restart is performed until $|F_k| \geq 1000$. Together with $\gamma = 100$, this means that we may only restart trees with $\hat{r}(k) \geq 10^5$ nodes. Further, after 1000 nodes, SCIP can use the branching history and other relevant data of a larger amount of solving nodes to build a better tree after the restart.

MIP Solver Experiments

We test our restart strategy with four different tree-size estimates: WBE; linear, a linear forecast of the search progress using double exponential smoothing with parameters $\alpha = \beta = 0.15$; window-acc and window-vel, a moving window with or without acceleration. Both use a window size of 100. Restarts at the root node are disabled. In order to limit the number of necessary experiments, we continue the solution process after the first clairvoyant restart without further interruptions. We compare to three other SCIP settings: default (default), no restarts (0-restart), and at most one restart at the end of the root node (1-restart) based on SCIP's default strategy.

We use SCIP 6.0 with SoPlex 4.0 as LP solver on the test set MMMc. All experiments have been conducted on a cluster with 48 nodes equipped with Intel Xeon Gold 5122 at 3.60GHz and 96GB RAM. Jobs were run exclusively on a node. The time limit was 2h.

Table 3 shows the solving time $\mathcal{T}$ in seconds and the number of solving nodes $\mathcal{N}$. If a restart was performed, $\mathcal{N}$ is

the sum of explored nodes in both search trees. The individual numbers are aggregated by a *shifted geometric mean* (Achterberg 2007b), with a shift of 1 for time and 100 for nodes. Two further columns $\mathcal{T}_{\text{rel}}$ and $\mathcal{N}_{\text{rel}}$ give the relative performance compared to `default`. Surprisingly, on the selected benchmark, `0-restart` is actually consistently better than `default` in terms of the solving time and solving nodes. The most likely explanation is that the default restart parameters were more efficient in a previous version of SCIP, but have not been completely re-calibrated in 6.0.

Group	Settings	#Restarts	$\mathcal{T}$	$\mathcal{T}_{\text{rel}}$	$\mathcal{N}$	$\mathcal{N}_{\text{rel}}$
ALL (490)	default	149	183.1	1.00	3 207	1.00
	0-restart	0	179.5	0.98	3 133	0.98
	1-restart	95	183.3	1.00	3 218	1.00
	WBE	27	179.9	0.98	3 139	0.98
	linear	168	183.6	1.00	3 185	0.99
	window-acc	103	175.6	0.96	3 016	**0.94**
	window-vel	147	178.2	0.97	3 101	0.97
AFF. (235)	default	149	414.0	1.00	26 767	1.00
	0-restart	0	397.3	0.96	25 533	0.95
	1-restart	95	414.1	1.00	26 956	1.01
	WBE	27	399.2	0.96	25 626	0.96
	linear	168	416.8	1.01	26 403	0.99
	window-acc	103	379.5	**0.92**	23 656	**0.88**
	window-vel	147	391.8	**0.95**	25 007	**0.93**
LT1000 (297)	default	123	21.2	1.00	520	1.00
	0-restart	0	20.6	0.97	509	0.98
	1-restart	75	21.1	1.00	525	1.01
	WBE	3	20.6	0.97	511	0.98
	linear	47	20.8	0.98	508	0.98
	window-acc	12	20.2	0.95	494	**0.95**
	window-vel	31	20.3	0.96	497	0.96
GE1000 (64)	default	18	2 088.7	1.00	33 209	1.00
	0-restart	0	2 046.7	0.98	31 384	**0.95**
	1-restart	14	2 127.2	1.02	33 760	1.02
	WBE	4	2 051.6	0.98	31 794	0.96
	linear	46	2 319.9	1.11	37 017	1.11
	window-acc	24	1 869.9	**0.90**	28 542	**0.86**
	window-vel	42	2 040.4	0.98	33 339	1.00

Table 3: Results for the proposed restart strategies within SCIP. Relative improvements of more than 5 % are indicated in **bold font**.

The table shows the results for 4 interesting groups of instances. The first group "ALL (490)" is the entire benchmark of 496 instances, reduced by 4 instances for which one of the versions reported a wrong result (an infeasible solution or a wrong bound). In addition, 2 instances were removed because of outstanding performance variability (Lodi and Tramontani 2013) unrelated to restarts.

The window methods achieve the best results in this experiment. The `window-acc` method yields a relative improvement of 4.1 % (time) and 5.9 % (nodes), followed by `window-vel` with 2.6 % (time) and 4.3 % (nodes), and `0-restart` with ≈ 2 % (time and nodes). Allowing only one restart does not show relevant differences compared to `default`, although `default` performs more than 50 additional restarts. From the column #Restarts, it is clear that not all instances are affected by restarts. The second group, "AFF. (235)", is therefore restricted to those 235 instances for which at least one of the settings performs a restart. As all proposed methods require at most a constant overhead per node, we do not show the group of unaffected instances.

The last two groups split the set of instances solved by at least one setting into an easier group for which all tested variants required at most 1000 seconds ("LT1000 (297)"), and a hard group for which at least one setting needed more than 1000 seconds. For the group "LT1000 (297)", `default` is the slowest among the tested settings. It is interesting to see that both `default` and `1-restart` trigger a large fraction of their total restarts on this easier group, which would be solved faster with fewer restarts, as shown by `0-restart`. The `window-acc` method only rarely performs a restart on this group, despite its size comprising 60 % of the total instance set. On the smaller (≈ 13 %) set of 64 harder, but solvable instances, `window-acc` performs more restarts than `1-restart` and even `default`, and reduces the solving time and nodes required by 10.5 % and 14 %, respectively.

Overall, the clairvoyant method `window-acc` achieves a considerable speedup compared to all available restart strategies that SCIP currently provides. The speedup is particularly high on the last set of very hard instances, where the use of acceleration shows a clear benefit compared to `window-vel`. Interestingly, the highest number of restarts is triggered by the clairvoyant restart using a linear forecast, while the backtrack estimator is by far the most conservative clairvoyant restart in this experiment. It appears that `window-acc` finds the best balance in that it neither restarts too often, nor too conservatively.

Conclusion and Future Directions

Although we have obtained performance improvements that are significant for MIP, it seems clear that we have only just begun tapping into the potential of clairvoyant algorithms. Undoubtedly, the reader themself is already envisioning a host of potential improvements and adaptations of this method, and we certainly recognize that all aspects, from the progress measure to the restarts, could be improved or extended. On the other hand, the simplicity of our method only better demonstrates the potential of this approach.

One notable improvement to clairvoyant restarts could consist in performing a sequence of restarts, which has proven beneficial in particular in CP and SAT.

Besides restarts, many other algorithmic aspects of MIP solvers could benefit from a tree-size estimate. For instance, after a clairvoyant restart, more cutting planes could be generated at the root node if branching has proven less efficient at reducing the gap than cuts have.

Finally, besides MIP, we anticipate that clairvoyant algorithms may lead to improvements in many other fields.

Acknowledgments

References

Achterberg, T., and Berthold, T. 2009. Hybrid branching. In Hoeve, W., and Hooker, J. N., eds., *Integration of AI and OR*

Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 5547 of *Lecture Notes in Computer Science*. Springer. 309–311.

Achterberg, T.; Koch, T.; and Martin, A. 2006. MIPLIB 2003. *Operations Research Letters* 34(4):361–372.

Achterberg, T. 2007a. Conflict analysis in mixed integer programming. *Discrete Optimization* 4(1):4 – 20.

Achterberg, T. 2007b. *Constraint Integer Programming*. Ph.D. Dissertation, Technische Universität Berlin.

Belov, G.; Esler, S.; Fernando, D.; Le Bodic, P.; and Nemhauser, G. L. 2017. Estimating the size of search trees by sampling with domain knowledge. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 473–479.

Berthold, T.; Hendel, G.; and Koch, T. 2017. From feasibility to improvement to proof: three phases of solving mixed-integer programs. *Optimization Methods and Software* 33(3):499–517.

Bixby, R. E.; Ceria, S.; McZeal, C. M.; and Savelsbergh, M. W. P. 1998. An updated mixed integer programming library: MIPLIB 3.0. *Optima* 58:12–15.

Chen, P. C. 1992. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing* 21(2):295–315.

2010. COR@L MIP Instances. http://coral.ie.lehigh.edu/data-sets/mixed-integer-instances/.

Cornuéjols, G.; Karamanov, M.; and Li, Y. 2006. Early estimates of the size of branch-and-bound trees. *INFORMS Journal on Computing* 18:86–96.

Gleixner, A.; Bastubbe, M.; Eifler, L.; Gally, T.; Gamrath, G.; Gottwald, R. L.; Hendel, G.; Hojny, C.; Koch, T.; Lübbecke, M. E.; Maher, S. J.; Miltenberger, M.; Müller, B.; Pfetsch, M. E.; Puchert, C.; Rehfeldt, D.; Schlösser, F.; Schubert, C.; Serrano, F.; Shinano, Y.; Viernickel, J. M.; Walter, M.; Wegscheider, F.; Witt, J. T.; and Witzig, J. 2018. The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin.

Kilby, P.; Slaney, J.; Thiébaux, S.; and Walsh, T. 2006. Estimating search tree size. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, 1014–1019. AAAI Press.

Knuth, D. E. 1975. Estimating the efficiency of backtrack programs. *Mathematics of Computation* 29:122–136.

Koch, T.; Achterberg, T.; Andersen, E.; Bastert, O.; Berthold, T.; Bixby, R. E.; Danna, E.; Gamrath, G.; Gleixner, A. M.; Heinz, S.; Lodi, A.; Mittelmann, H.; Ralphs, T.; Salvagnin, D.; Steffy, D. E.; and Wolter, K. 2011. MIPLIB 2010. *Mathematical Programming Computation* 3(2):103–163.

Kullmann, O. 2009. Fundaments of branching heuristics. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 205–244.

Le Bodic, P., and Nemhauser, G. 2017. An abstract model

for branching and its application to mixed integer programming. *Mathematical Programming* 166(1):369–405.

Lelis, L. H. S.; Otten, L.; and Dechter, R. 2013. Predicting the size of depth-first branch and bound search trees. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 594–600. AAAI Press.

Lobjois, L., and Lemaître, M. 1998. Branch and bound algorithm selection by performance prediction. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, 353–358. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Lodi, A., and Tramontani, A. 2013. *Performance Variability in Mixed-Integer Programming*. chapter 2, 1–12.

Özaltın, O. Y.; Hunsaker, B.; and Schaefer, A. J. 2011. Predicting the solution time of branch-and-bound algorithms for mixed-integer programs. *INFORMS Journal on Computing* 23(3):392–403.

Purdom, P. W. 1978. Tree size by partial backtracking. *SIAM Journal on Computing* 7(4):481–491.

Thayer, J. T.; Stern, R.; and Lelis, L. H. S. 2012. Are we there yet? - estimating search progress. In Borrajo, D.; Felner, A.; Korf, R. E.; Likhachev, M.; López, C. L.; Ruml, W.; and Sturtevant, N. R., eds., *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press.