Early Estimates of the Size of Branch-and-Bound Trees

Gérard Cornuéjols[*], Miroslav Karamanov[†], Yanjun Li[‡]

23th April 2004

Abstract

This paper intends to show that the time needed to solve mixed integer programming problems by branch and bound can be roughly predicted early in the solution process. We construct a procedure that can be implemented as part of an MIP solver. It is based on analyzing the partial tree resulting from running the algorithm for a short period of time, and predicting the shape of the whole tree. The procedure is tested on instances from the literature. This work was inspired by the practical applicability of such a result.

1. Introduction

The effectiveness of the branch-and-bound procedure for solving mixed integer programming (MIP) problems has made it a method of choice in commercial software for several decades. Its applicability to large instances has increased in the last ten years with the increased computational power of computers as well as substantial improvements in algorithms. Although current software packages are able to solve many large instances by branch and bound and its modifications, there are also many other instances where they fail due to the excessive size of the enumeration tree.

The branch-and-bound algorithm is a divide-and-conquer approach that dynamically constructs a search tree, each node of which represents a subproblem. Upper and lower bounds can be obtained from feasible solutions and from solving the linear programming relaxation of these subproblems. These bounds are used to prune the tree. In addition to the bounds, the search strategies determine the size and shape of the search tree. Good descriptions of the branch-and-bound algorithm can be found in Nemhauser and Wolsey (1988), and Wolsey (1998). An extensive study of search techniques is presented in Linderoth and Savelsbergh (1997).

The application of the branch-and-bound algorithm can be limited by both the computing time and the storage space required (even when storing nodes on a hard disk). The solution process may take hours or days and there is very little a priori indication of how difficult a

[*]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 15213, and LIF, Faculté des Sciences de Luminy, 13288 Marseille, France, gc0v@andrew.cmu.edu

[†]Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA, 15213, miroslav@andrew.cmu.edu

[‡]Krannert Graduate School of Management, Purdue University, West Lafayette, IN 47907, li14@mgmt.purdue.edu

model will be to solve. Unfortunately, there is no known method to extract this information from the problem formulation. Practice shows that even small modifications in a model can increase or decrease the solution time by an order of magnitude. On the other hand, many present day applications require a solution of MIP problems within minutes. Some specialized commercial software products for solving MIP problems apply only heuristics because speed is more important than obtaining an optimal solution. Therefore, from a practical point of view, even a rough estimate of the computating time required by branch and bound would be useful. It can help decide whether to continue with branch and bound or switch to heuristics.

Memory requirements are also critical. To store the tree may require enormous space and it is possible for the branch-and-bound algorithm to terminate prematurely after many hours of work without providing a satisfactory solution due to a lack of memory. For example, some instances from MIPLIB, a standard library of test problems, require many gigabytes for node storage.

The CPU time required for a branch-and-bound solution depends roughly linearly on the number of nodes in the branch-and-bound tree (for simplicity, we will call it *the tree*). In the present paper we attempt to devise a method for estimating the total size of the tree at an early stage of the solution process. We define the following requirements for the method:

- It should function as part of a general-purpose MIP solver. It should provide predictions without controlling or directing the solution process.
- It should be able to output a prediction with satisfactory precision after a short period of time (e.g., five seconds for medium size problems). It should be able to update the prediction as time elapses.
- The additional computations for these predictions should consume a negligible amount of time compared to the branch-and-bound algorithm. They should not slow down the solution process. They should rely as much as possible on the data obtained from the MIP solver.

"Satisfactory precision" can be defined in different ways. We propose to measure the precision by the *error factor*—the factor by which the prediction under- or overestimates the actual tree size. We consider that a prediction within an error factor of five provided after five seconds of solution time is satisfactory. Such a prediction will allow us to conclude whether the solution will take minutes, hours, or days. For example, an estimated solution time of 1 hour would be interpreted as saying that the instance can be solved between 12 minutes and 5 hours. If we had set a time limit of 10 hours and the actual solution time of an instance exceeded 10 hours, the 1-hour prediction would not be considered satisfactory, whereas a 4-hour prediction would be.

We introduce a notion related to the shape and size of a tree called γ-sequence. The nodes at distance i from the root node are said to occupy level i. Let the *width* of a level be the number of nodes at that level. We define γ_i as the ratio between the width of level $i+1$ and that of level i. The γ-sequence of a tree is the sequence of γ_i, for all levels i with positive width. Given the γ-sequence of a tree, we can reproduce the number of nodes at each level. Our main goal is to obtain a satisfactory approximation to the γ-sequence. After running the branch-and-bound algorithm for a short period of time, we obtain a subtree of the whole branch-and-bound tree. One approach is to use the γ-sequence of the partial tree as a basis for the estimation. Our tests showed that this does not lead to good results. Instead, we will use the partial tree to estimate three key parameters of the complete tree: the depth, the last full level, and the waist level. We will use these parameters for modeling the γ-sequence. We describe and analyze our approach and present our computational results in Section 3.

In 1975 Knuth proposed a procedure for estimating the size of branch-and-bound trees based on sampling by random paths. Discussion of this method and its application is included in Section 2.

The testbed for the experiments described in this paper includes 28 instances from MIPLIB 3.0 (Bixby et at. 1998). The solution of most of them requires building a branch-and-bound tree of more than 1000 nodes. We also included some of the "smaller" instances. The choice is aimed at obtaining diversity while concentrating on the nontrivial instances. We also tested our prediction algorithm on additional instances from the literature. The computations were made on a Sun Ultra 60 (360MHz UltraSPARC-II processor) with ILOG CPLEX 8.0. Because our goal in this paper focuses on the branch-and-bound algorithm, we did not apply heuristics and cuts at nodes other than the root node, except at the very end of Section 3 where we briefly report on our experience with branch and cut.

2. Earlier work

Knuth (1975) was the first to discuss how to estimate the size of a general backtrack tree. The method that he proposed is a random exploration of the tree based on a Monte Carlo approach. The algorithm repeatedly traverses random paths from the root node to the leaves, without backtracking. At each node, one of its successors is chosen at random according to a uniform probability. The estimate of the number of nodes in the tree is the average over several runs of $1 + d_1 + d_1 d_2 + \ldots + \prod_{i=1}^{k} d_i$, where d_i is the number of successors to the chosen node at level i and k is the depth reached. Furthermore, Knuth generalized the above simple, unbiased method to allow the selection of random paths under non-uniform probabilities. He proves that the expected value of both estimates, unbiased and biased, is the size of the search tree, and he provides upper bounds on the variance of the estimates.

Knuth's algorithm has been improved in various ways by Purdom (1978) and Chen (1992). The modified algorithm by Purdom attempts to reduce the variance of the estimate by allowing more than one branch out of a node to be further investigated. Chen adopted a stratified sampling approach, based on a "heuristic function" (stratifier) supplied by the algorithm designer. Chen proved that, by exploiting the tree structure reflected by the stratifier, the heuristic sampling method reduces the variance relative to Knuth's algorithm.

Although he did not present many test results, Knuth provided a good insight into the potential problems that may arise when applying his procedure. He emphasized the large variance of the estimator, as well as the tendency to get underestimations when the deep levels are visited with very low probability. Another important remark was that "the estimation procedure does not apply directly to branch-and-bound algorithms," unless the optimal objective value is given a priori as a bound. Thus, the procedure can be used to estimate the amount of work to prove any given bound for optimality.

Nevertheless, Knuth's method has been employed for estimating the size of a branch-and-bound tree. Lobjois and Lemaitre (1998) proposed a method to select, for each instance of the maximal constraint satisfaction problem, the most appropriate branch-and-bound algorithm from among several candidates. They compared the running times of the algorithms predicted by Knuth's procedure and concluded that, despite the great variability of the estimates, it selects the best algorithm in most cases. One conclusion was that Knuth's estimator can be used for comparison purpose even when it outputs imprecise predictions.

Brüngger et al. (1998) set up such an estimator in their solver to predict the running time when tackling large-scale quadratic assignment problems (QAP) by parallel computation. Anstreicher et al. (2002) also used Knuth's procedure for estimating the solution time of a specialized branch-and-bound algorithm for QAP. They reported excellent results of the basic, unbiased method for instances of size less than 24 but pointed out that the quality of the estimation rapidly deteriorates as the size of the problems increased. To fix this, they applied "importance sampling", identical to the biased sampling proposed by Knuth. They suggested the use of non-uniform probabilities that depend on the inherited relative gap at a node. In addition, they proposed a way of reducing the variance of the estimates at deeper levels in the tree and avoiding wasteful duplication of computations at low levels. Rather than start the random dives at the root node, they first ran the branch-and-bound algorithm in breadth-first mode to obtain all nodes at a predetermined level, and then initialized Knuth's algorithm from a node at that level. The modified procedure output very good estimations for QAP problems of up to size 30.

We performed tests with our set of MIPLIB instances but could not observe the good estimation properties of Knuth's procedure reported in the aforementioned papers. We applied unbiased random sampling with 1000 iterations. Even when the optimal objective value was provided as a cutoff bound, the error factor of the prediction was greater than 5 in 11 of the 23 instances solved to optimality (cf. Table 1). Five instances were not solved within 10 hours of computing time and 1GB of storage space. For these instances, a correct prediction should exceed the number of nodes at interruption. Knuth's method provided such a prediction in three cases and produced significant underestimations in the other two cases. The large number of errors can be attributed to the large variance of the estimator and to the insufficient number of iterations (1000 while Anstreicher et al. proposed 10,000). In addition, even with this relatively small number of samples, the estimation time was significant. It was greater than one minute for all problems but one, and it was greater than five minutes in 10 out of 28 cases. For many of the instances, this is an unreasonably long period of time devoted solely to time estimation without contributing to the solution. Moreover, in 11 cases, the number of nodes visited during the estimation procedure exceeded the size of the branch-and-bound tree, i.e., the estimation procedure took longer to execute than the solution algorithm itself. Even with this abundant information, the prediction error factor was greater than five in five of these 11 cases. When we applied 10,000 samples in order to obtain a more precise estimate, the estimation time became longer than the solution time for 20 out of 28 instances. This made the price for the increased precision too high.

Furthermore, starting with information about the optimal objective value is not realistic. The above experiment, repeated with no cutoff bound, lead to huge overestimations (by factors of 10^2 to 10^{31}) for almost all problems while the estimation time was even longer than that reported in Table 1.

We can see three main reasons for the observed inaccuracy. First, the lack of a good cutoff bound results in a very small amount of pruning in the second experiment. Second, due to the exponential growth of the estimate with node depth, the error tends to be small when the tree is shallow, as those studied by Anstreicher et al., but when the tree is deep, e.g., more than 100 levels, even one or two sample paths that go to the deepest levels can cause a huge overestimation. Third, it is possible that Knuth's procedure works much better in some classes of problems and with some types of branch-and-bound algorithms than with others. (The algorithm employed by Anstreicher et al. is specialized for QAP.)

Our conclusion is that in most cases Knuth's method is not practical for early prediction of the solution time of general MIP problems. We would like to have a much faster routine with acceptable precision that does not assume prior knowledge of the optimal objective value.

Our exploration in this paper is distinct from Knuth's work in the following four respects.

Table 1: Tree size estimation by Knuth's method with sample size 1000

Problem	Predicted number of nodes	Actual number of nodes	Ratio	Estimation Time [seconds]	Estimation Nodes visited
air05	754	1221	0.61	31200	5690
arki001 *	262	1124575	2.3E-4	2838	4695
bell3a	14110	18512	0.76	121	13736
bell4	1.3E+06	13654	95	78	9261
bell5	1362	301146	0.004	61	4451
blend2	365	5750	0.063	201	6683
gesa2_o	6011	1136	5.2	809	11187
harp2 *	22775	786616	0.028	925	9563
lseu	4738	1614	2.9	41	9578
markshare1 *	1.2E+10	52464676	228	94	30855
markshare2 *	5.0E+12	45059758	1.1E+5	129	38200
mas74	8.7E+06	10159496	0.85	182	20251
mas76	1.0E+06	637057	1.6	144	16655
misc07	21697	111784	0.19	1289	10580
mod008	1373	2161	0.63	92	13377
mod011	25945	21788	1.2	12180	9683
modglob	2.0E+06	304	6480	209	11804
noswot	4.1E+07	5614491	7.3	112	13266
pk1	163508	337940	0.48	155	14529
pp08a	9.5E+09	933	1.0E+7	204	28249
pp08aCUTS	4.6E+06	1687	2744	360	19624
qiu	40163	9358	4.3	6081	11556
rgn	1261	3025	0.42	73	9665
rout	34871	1797969	0.019	1643	11024
seymour *	4.4E+15	54713	8.0E+10	128520	42574
stein27	8429	3706	2.3	200	12573
stein45	167466	68093	2.5	889	15508
vpm2	337552	25255	13	216	14180

* Solution procedure interrupted. Column "Actual number of nodes" contains the number of nodes at time of interruption.

First, it does not rely on an initial bound, although having such a bound would be advantageous. Second, our procedure employs a standard branch-and-bound algorithm, which does backtracking and updates the bound. Third, our method is based on estimating parameters of the enumeration tree and extrapolating its γ-sequence from these parameters, rather than estimating the γ-sequence directly by the number of descendants. Finally, our estimation procedure analyzes the partial tree produced by the branch-and-bound algorithm and then continues the search. Therefore, the work done in the estimation phase is essentially the beginning of the solution process. While the random sampling can find a good solution by chance, the time spent by this sampling procedure is usually lost for the solution of the problem.

Some other means of estimating the termination time of a branch-and-bound algorithm have been considered as well. One idea is to estimate upper and lower bounds on the objective value as a function of time and then apply simple regression. A ballpark estimate of running time can be obtained by extrapolating those curves and predicting when the gap will be zero. One could also consider the number of active nodes in the queue as a function of time and, again, extrapolate the curve. Both approaches require a significant amount of solution time in order to capture the trend. The gap closes in large steps at the beginning and in much smaller ones later on. The behavior of the set of active nodes is very problem specific. Our experience shows that the gap closed in the first 5–10 seconds and the dynamics of the set of active nodes in the same period hardly provide sufficient information to make a sensible prediction. Howerer, combining all these methods could lead to a more precise estimator. Further investigation in this direction should be fruitful.

3. Our method

3.1. General description

In what follows, we assume that the maximum number of descendants of a node is two. By redefining the last full level, our estimation procedure can accommodate a branching scheme with any number of descendants.

Definition 1 *In a branch-and-bound tree T, let $w_T(i)$ be the* width *of level i, i.e., the number of nodes at that level. Let $d_T = max\{i : w_T(i) > 0\}$ be the* depth *of the tree. Level $l_T = \min\{i : \frac{w_T(i+1)}{w_T(i)} < 2,\ 0 \le i \le d_T\}$ is called the* last full level *of the tree (assuming that each node has at most two successors). Up to this level, the tree is a complete binary tree. Let the* waist *of the tree be the level with maximum width, $b_T = \arg\max\{w_T(i) : 0 \le i \le d_T\}$. When this level is not unique, define $b_T = \left\lceil \frac{b_1+b_2}{2} \right\rceil$, where $b_1 = \min\{i : w_T(i) = t\}$, $b_2 = \max\{i : w_T(i) = t\}$, and $t = \max\{w_T(i) : 0 \le i \le d_T\}$, i.e., b_T is the center of the smallest interval containing all the levels with maximum width. Let $n(T) = \sum_{i=0}^{d_T} w_T(i)$ be the number of nodes in T. The sequence $\{w_T(i) : 0 \le i \le d_T\}$ is called the* profile *of tree T.*

A framework for estimating the size of a branching tree T is given as follows:

```
Input: A mixed integer programming problem
Initialization. Set counters for the width of the levels,
    $w(i) = 0$, for $i = 0, \dots, \widetilde{D}$, and $\widetilde{D}$ large enough.
Step 0. Run the branch-and-bound algorithm.
Step 1. At each node of the branching tree, increment the
    corresponding counter by 1.
Step 2. Stop when a prespecified event occurs (e.g., a time
    or node bound is reached), and let $t$ be the resulting
    subtree of $T$.  Set $w_t(i) = w(i)$, $\forall i$.
Step 3. Find $l_t$, $b_t$, and $d_t$.
Step 4. Construct a measurement tree $M$.
Step 5. If the problem is not solved, output $n(M)$ (an
    estimate of $n(T)$), continue the branch-and-bound
    algorithm, and go to Step 1.  Otherwise, terminate.
```

The branch-and-bound algorithm is paused at a given point in time. The resulting tree of visited nodes, t, also called *partial tree*, is used to estimate the parameters of the complete branch-and-bound tree. In Step 3 we find the last full level, the waist, and the depth of the partial tree, which serve as estimates of the parameters of the complete tree. The *measurement tree* constructed in Step 4 is not a real tree but a profile of a tree that is designed to replicate, as much as possible, the profile of the estimated tree. It is built according to a model to be discussed in the next section. The number of nodes in the measurement tree is used as an estimate of the total number of nodes in the branch-and-bound tree.

We apply this procedure repeatedly in order to output periodic estimations until the branch-and bound algorithm terminates. We formally divide the solution process into two phases. Phase I ends with the output of the first prediction. In Phase II, we output periodic predictions. As a termination criterion for Phase I, we require that both of the following conditions be satisfied: the solution time is at least 5 seconds and the number of nodes in the partial tree is at least 20 times the depth of the partial tree ($n(t) \geq 20d_t$). The second condition is important because a reasonable level width is necessary in order to obtain a sensible approximation of the parameters of the complete tree. The factor of 20 is established empirically and can be changed to reflect tradeoffs between speed and accuracy of the first prediction.

In the above procedure, branch and cut can be used instead of branch and bound. It is important to note that cuts added after branching has started can change the structure of the branching tree and affect the validity of the predictions.

3.2. The linear model for estimating the γ-sequence

In this section, we describe a model for the measurement tree needed in Step 4 of the above procedure. We propose to model the profile of the complete tree using three parameters only, namely l_t, b_t, and d_t.

A characteristic that uniquely defines a tree profile is its γ-sequence—the ratios that describe the change of width from one level to the next.

Definition 2 *Consider a branch-and-bound tree T and let d_T be its depth. The sequence $\gamma_0, \gamma_1, \dots, \gamma_{d_T}$ is called the γ-sequence of this branch-and-bound tree, where $\gamma_i = \frac{w_T(i+1)}{w_T(i)}$, for $0 \leq i \leq d_T$.*

Given the γ-sequence of a tree T, the width of a particular level i is $w_i = \prod_{j=0}^{i-1} \gamma_j$. The size of the tree is $n(T) = 1 + \sum_{i=1}^{d_T} \prod_{j=0}^{i-1} \gamma_j$. A tree model is essentially a model for building the γ-sequence.

We analyzed the profiles of branch-and-bound trees obtained using the CPLEX default solution settings. The solution algorithm was cut and branch, where cuts are applied only at the root node. Our tests show that, for almost all of the problems in MIPLIB, the profile of the tree looks like a bell-shaped curve. The same observation is made by Knuth (1975) for backtrack algorithms in general. The γ-sequence is generally decreasing for i greater than the last full level and the value of γ_i is approximately 1 at the waist and 0 at the deeepest level. This observation justifies the use of a linear model for the change of γ, defined by the formula:

$$\gamma_i = \begin{cases} 2, & \text{for } 0 \leq i \leq l_T - 1, \\ 2 - \frac{i - l_T + 1}{b_T - l_T + 1}, & \text{for } l_T \leq i \leq b_T - 1, \\ 1 - \frac{i - b_T + 1}{d_T - b_T + 1}, & \text{for } b_T \leq i \leq d_T. \end{cases}$$

This simple model outputs satisfactory estimations in the majority of the cases. Figure 1 shows a typical tree profile (the solid line) and the measurement tree obtained by the linear tree model (the dashed line). The proximity of the two lines is common for most problems with a bell-shaped tree profile.

The estimation properties of the linear model are tested in experiments with the set of 28 problems from MIPLIB. In this part, we assume that the exact values of l_T, b_T, and d_T for the complete trees are available and we study the accuracy of the tree model.

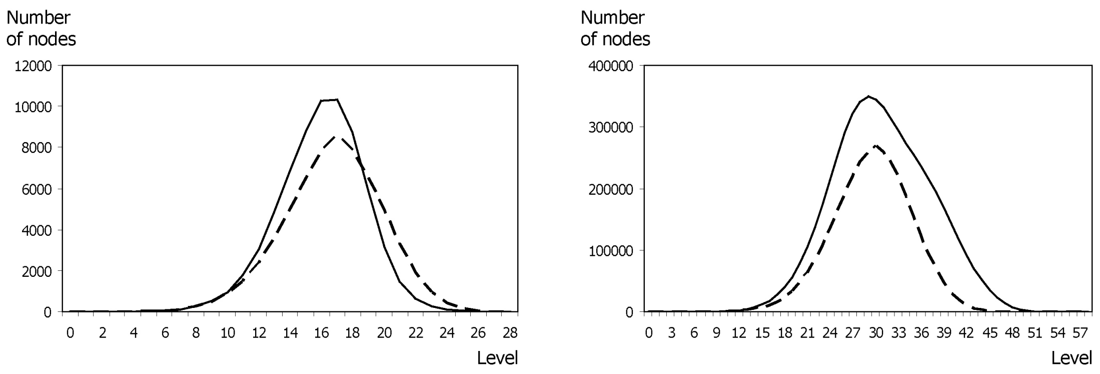

Figure 1: Profiles of the actual tree and linear model measurement tree. Problems: stein45 and noswot.

Table 2: Linear model estimation

Problem	Actual number of nodes	Linear model estimation	Ratio
air05	1221	3017	2.47
arki001 *	1124575	1910566720	1698.92
bell3a	18512	3217	0.17
bell4	13654	20639	1.51
bell5	301146	47977699	159.32
blend2	5750	13401	2.33
gesa2_o	1136	6325	5.57
harp2 *	786616	261277428	332.15
lseu	1614	3023	1.87
markshare1 *	52464676	20885162	0.40
markshare2 *	45059758	925206188	20.53
mas74	10159496	6257366	0.62
mas76	637057	703329	1.10
misc07	111784	6657013	59.55
mod008	2161	39000	18.05
mod011	21788	41535	1.91
modglob	304	472	1.55
noswot	5614491	3336543	0.59
pk1	337940	1532758	4.54
pp08a	933	1171	1.26
pp08aCUTS	1687	3472	2.06
qiu	9358	60458	6.46
rgn	3025	19660	6.50
rout	1797969	27316720	15.19
seymour *	54713	72319299	1321.79
stein27	3706	3996	1.08
stein45	68093	63255	0.93
vpm2	25255	31299	1.24

* Problem not solved to completion.

Table 2 compares the size of the measurement tree obtained by the linear model with the actual number of nodes in T. The last column shows the ratio between the two.

The solution of the problems marked with an asterisk has been interrupted after ten hours and, therefore, the figure in the second column is the size of the branching tree at interruption. We analyze this group of problems separately. A desirable output of the tree model for these instances is an estimation greater than the number of nodes processed before interruption. This is observed in four out of five cases. In the fifth case, the ratio between the predicted number of nodes and the number of nodes after ten hours of computation is 0.4. In this paper, we consider this satisfactory (i.e., within a factor of 5) although we do not know the true ratio between the predicted and the actual number of nodes.

Twenty three problems were solved to completion. For fifteen problems, the error is within a factor of five. For eleven of them the error is within a factor of two. Our conclusion is that the linear model provides a satisfactory estimation of the actual tree profile for most instances and, therefore, can be used to estimate the number of nodes in the tree.

3.3. Computational experience (MIPLIB instances)

We performed computational experiments with the procedure described above and our testbed of 28 MIPLIB problems. We ran the branch-and-cut algorithm of CPLEX 8.0 with its default branching and node selection rules but with the restriction that cuts and heuristics were applied only at the root node. The results are presented in Table 3. The solution process has been interrupted when solution time exceeded 10 hours or when the branching tree size exceeded 1GB. These instances are marked with an asterisk. We applied the linear tree model for the construction of the measurement tree based on the parameters l_t, b_t, and d_t of the partial tree obtained at the end of Phase I. The predicted tree size is reported in the second column of Table 3. The third column contains the actual tree size and the ratio between the prediction and the true value is shown in the fourth column. The fifth column contains the time to obtain the prediction.

We compute a time estimate for solving the instance. This time estimate θ equals the size of the measurement tree times the average solution time of a node in the partial tree, based on the assumption that the average running time at a node is relatively constant during the solution. Instead of the point estimate θ, we output a range $[\alpha, \beta]$ for the solution time, shown in column six. The width of this range corresponds to an estimation error of five. Specifically, $\alpha = \max\{\text{Phase I time}, 0.2\theta\}$, $\beta = 5\theta$ (or $+\infty$ when $5\theta > 10$ hours), and we round seconds and minutes to the nearest multiple of five, and hours (or minutes smaller than five) to the nearest integer. The true solution time or the time until interruption is given in the last column. Incorrect predictions are marked by a dagger. For instances solved to completion, a dagger marks the cases for which our prediction interval does not contain the actual solution time. For the instances that are not solved to completion, a dagger marks the cases where we predicted that the instance could be solved in less than 10 hours. Note that, when an instance was interrupted because of space limitation, it could happen that our time prediction is within a factor of 5 of the actual unknown solution time but we still consider these to be incorrect predictions. For example, if the predicted solution time is between 15 minutes and 6 hours and the solution process is interrupted after 2 hours because of space limitation, we consider the prediction incorrect.

Five instances, lseu, mod008, modglob, rgn, and stein27, were solved during Phase I. They are not present in the table. Eighteen of the remaining instances were solved to optimality. For ten of them the prediction is correct. For two instances, bell3a and gesa2_o, the error is small. There are four cases, mas74, misc07, noswot, and rout, with a considerable error. The solution

Table 3: Predictions by our estimation procedure

Problem	Predicted number of nodes	Actual number of nodes	Ratio	Phase I time [s]	Predicted time range	Actual solution time	
air05	2043	1221	1.7	291	5 m – 40 m	5 m	
arki001 *	26808093	1124575	23	61	> 10 h	> 10 h	
bell3a	3217	18512	0.17	5	5 s – 15 s	20.7 s	†
bell4	13980	13654	1	5	5 s – 1 m	12.2 s	
bell5	987629	301146	3.2	5	2 m – 1 h	4.2 m	
blend2	59796	5750	10	7	40 s – 15 m	15.5 s	†
gesa2_o	6325	1136	5.6	8	10 s – 5 m	8.4 s	†
harp2 *	3644539	786616	4.6	45	> 1 h	> 1.2 h	
markshare1 *	13480899	52464676	0.26	5	> 30 m	> 10 h	
markshare2 *	925206188	45059758	21	5	> 10 h	> 10 h	
mas74	118255	10159496	0.01	5	30 s – 15 m	3.9 h	†
mas76	69963	637057	0.11	5	15 s – 10 m	10.0 m	
misc07	1437454696	111784	13000	14	> 10 h	10.7 m	†
mod011	9576	21788	0.44	429	15 m – 6 h	2.0 h	
noswot	74866	5614491	0.01	5	25 s – 10 m	2.2 h	†
pk1	31044	337940	0.09	5	10 s – 5 m	9.4 m	†
pp08a	786	933	0.84	5	5 s – 25 s	5.6 s	
pp08aCUTS	2355	1687	1.4	5	5 s – 1 m	9.1 s	
qiu	2085	9358	0.22	65	1 m – 20 m	9.5 m	
rout	8758	1797969	0.01	21	20 s – 10 m	2.9 h	†
seymour *	182659036	54713	3300	2171	> 10 h	> 10 h	
stein45	43822	68093	0.64	5	30 s – 10 m	2.2 m	
vpm2	9176	25255	0.36	5	5 s – 1 m	40.6 s	

* Instance not solved to completion.

† Incorrect time prediction.

of five instances was interrupted after exceeding the time or space limit, and this was predicted correctly. Overall, the results are satisfactory considering the great diversity of the instances.

3.4. Computational experience (additional instances)

Our procedure for estimating the number of nodes in a branch-and-bound tree was designed based on observations from a diverse sample of 28 instances from the MIPLIB. In order to validate the procedure, we applied it to an independent test set. We used MIP benchmarks from the literature representing several different problem types. The results are reported in Tables 4 and 5.

The first group of instances are multidimensional knapsack problems from Beasley (1990b) and Chu (1998). Due to a significant amount of pruning, the branch-and-bound trees tend to be slim and deep. Our procedure deals relatively well with the lower dimensional instances (mknapcb1, mknapcb4, mknapcb7) but not as well with higher dimensional problems. Overall, 13 out of 27 instances are solved to completion. Correct predictions are obtained for seven of them, for two instances the error in prediction is small, and in four cases the error is significant. The solution of 14 instances was interrupted, in most cases because of the space limit. For 11 of them, this interruption was predicted correctly.

Table 4: More test results

Problem	Phase I time [s]	Predicted time range	Solution time	
Multidimensional Knapsack Problems				
mknapcb1-1	5	5 s – 2 m	1.3 m	
mknapcb1-11	5	5 s – 20 s	7.7 s	
mknapcb1-21	5	5 s – 50 s	5.3 s	
mknapcb2-1	5	> 1 h	8.2 m	†
mknapcb2-11	5	> 10 h	20.1 m	†
mknapcb2-21	5	> 10 h	15.3 m	†
mknapcb3-1 *	15	> 10 h	> 2.6 h	
mknapcb3-11 *	15	> 10 h	> 2 h	
mknapcb3-21	16	> 10 h	6.4 m	†
mknapcb4-1	5	15 s – 10 m	13.4 m	†
mknapcb4-11	5	35 s – 15 m	5.9 m	
mknapcb4-21	5	5 s – 50 s	17.9 s	
mknapcb5-1 *	7	15 m – 6 h	> 2.5 h	†
mknapcb5-11 *	7	> 2 h	> 2.3 h	
mknapcb5-21 *	7	10 m – 4 h	> 2.6 h	†
mknapcb6-1 *	22	> 10 h	> 2.8 h	
mknapcb6-11 *	25	25 m – 10 h	> 2.9 h	†
mknapcb6-21 *	23	> 10 h	> 2.1 h	
mknapcb7-1	5	5 m – 1 h	23.2 m	
mknapcb7-11	5	10 m – 3 h	3.3 h	†
mknapcb7-21	5	5 m – 1 h	7.9 m	
mknapcb8-1 *	18	> 10 h	> 10 h	
mknapcb8-11 *	14	> 1 h	> 10 h	
mknapcb8-21 *	15	> 4 h	> 8.8 h	
mknapcb9-1 *	74	> 10 h	> 5.6 h	
mknapcb9-11 *	63	> 10 h	> 4.8 h	
mknapcb9-21 *	50	> 8 h	> 4.3 h	
Set Covering Problems				
scpnre1	581	10 m – 3 h	58.3 m	
scpnre2	654	> 30 m	7.6 h	
scpnre3	707	15 m – 7 h	1.1 h	
scpnre4	188	5 m – 1 h	37.1 m	
scpnre5	87	2 m – 40 m	21.7 m	
scpnrf1	1311	20 m – 7 h	29.7 m	
scpnrf2	860	15 m – 3 h	19.9 m	
scpnrf3	625	10 m – 3 h	13.6 m	
scpnrf4	635	15 m – 6 h	1.4 h	
scpnrf5	407	15 m – 7 h	2.1 h	
scpnrg1 *	1625	> 1 h	> 10 h	
scpnrg2 *	558	15 m – 5 h	> 10 h	†
scpnrg3 *	846	> 4 h	> 10 h	
scpnrg4 *	1166	> 10 h	> 10 h	
scpnrh1 *	3429	> 10 h	> 10 h	
scpclr10 *	106	> 10 h	> 10 h	
scpclr11 *	2203	> 10 h	> 10 h	
scpclr12 *	12501	> 10 h	> 10 h	

* Instance not solved to completion.

† Incorrect time prediction.

Table 5: More test results

Problem	Phase I time [s]	Predicted time	Solution time	
Bin Packing Problems				
t60_00 *	174	> 10 h	> 10 h	
t60_05 *	93	> 10 h	> 10 h	
t60_10 *	238	> 10 h	> 10 h	
t60_15 *	432	> 10 h	> 10 h	
t120_00 *	1487	> 10 h	> 10 h	
t120_05 *	7329	> 10 h	> 10 h	
t120_10 *	8125	> 10 h	> 10 h	
t120_15 *	1681	> 10 h	> 10 h	
u120_00 *	663	> 10 h	> 10 h	
u120_05 *	1010	> 10 h	> 10 h	
u120_10 *	1229	> 10 h	> 10 h	
u120_15 *	951	> 10 h	> 10 h	
Capacitated Facility Location Problems				
capa1 *	242	> 10 h	> 2.8 h	
capa2 *	241	> 10 h	> 2.8 h	
capa3 *	238	> 10 h	> 2.8 h	
capa4 *	237	> 10 h	> 2.8 h	
capb1 *	209	> 10 h	> 2.5 h	
capb2 *	209	> 10 h	> 2.5 h	
capb3 *	209	> 10 h	> 2.5 h	
capb4 *	209	> 10 h	> 2.5 h	
capc1 *	200	> 10 h	> 2.5 h	
capc2 *	201	> 10 h	> 2.5 h	
capc3 *	222	> 10 h	> 2.6 h	
capc4 *	219	> 10 h	> 2.6 h	
MIP Benchmarks				
bc *	1315	> 10 h	> 10 h	
binkar10_1 *	21	5 m – 3 h	> 6.4 h	†
eilD76	469	> 10 h	30.1 m	†
mas284	10	10 s – 5 m	1.2 m	
mkc1	159	> 10 h	4.3 h	†
prod1	5	1 m – 25 m	11.2 m	
ran14x18_1 *	24	> 10 h	> 10 h	

* Instance not solved to completion.

† Incorrect time prediction.

The second group consists of set covering instances from Beasley (1990a, 1990b). For these instances, the estimation requires typically more than one minute, in some cases more than ten minutes. The reason is the long solution time at a node (on average, 40 times longer than for the group of multidimensional knapsack problems). The estimation procedure performs well for these instances. There is only one incorrect prediction out of 18.

The third group of instances are bin packing problems from Falkenauer (1994), Beasley (1990b). The solution of all these instances takes more than 10 hours and our method provides correct estimations in all cases. Due to the long solution time of a single subproblem, the Phase I time is much longer than five seconds, reaching more than two hours in two cases. This may seem too long for a prediction but this is the time to make only about 20–30 dives in the tree. If Knuth's estimation method was applied with 1000 or 10,000 dives, the prediction procedure would hardly be practical.

The fourth type of instances we tested are capacitated facility location problems from Beasley (1988, 1990b). A huge solution tree is typical, which leads to exceeding the space limit after less than three hours in all of the cases. (The space limit applied to this group of instances was 3GB.) Interruption was predicted correctly for all these instances.

Finally, we tested seven other MIP benchmark instances from the web site of Argonne National Laboratory and H. Mittelmann's web site. The prediction is correct for four of them.

In this section, 76 additional instances were tested. Twenty seven of them were solved to completion. For these 27 instances, the number of correct predictions was 19. In two cases, the prediction is close to the actual solution time and in six cases the error of prediction is significant. The solution of 49 instances was interrupted due to the time or space limit. For them, there are only 5 cases of incorrect prediction.

Overall, we tested 99 MIP instances. The predicted time range was correct for 78 instances. (In this summary, we exclude the five MIPLIB problems that were solved during Phase I.) In particular, there were 54 instances that required excessive time (more than 10 hours) or space (more than 1GB) and this was predicted correctly in 49 cases. For the 45 instances that could be solved within the 10 hour and 1GB limits, this fact was predicted correctly for 39 of them. In other words, given the time and space limitations that we set for these experiments, the estimation procedure estimated correctly whether an instance could be solved in 88 out of 99 cases. We conclude that, although not precise, this method often provides a reasonable early estimate of the computing time of a branch-and-bound algorithm.

3.5. Analysis and refinements

In this section we identify several sources of imprecision in our estimation procedure and we discuss possible remedies. We also discuss our experience with branch and cut.

3.5.1. The linear model

The results reported in Section 3.2 show that for most problems, the linear model has satisfactory precision. However, it does not perform well when the tree is deep and slim. The problem stems from the fact that the linear model uses only three parameters, the last full level, the waist, and the depth. The model does not incorporate an estimate of the maximum width of the tree, which we might call the *waistline*, i.e., the width at the waist. If a slim tree and a fat tree have identical last full level, waist, and depth, the model will output the same estimation. The linear model assumes that γ decreases linearly from 2 to 1, which reproduces satisfactorily the profiles of most of the branching trees we tested. But if the actual decrease is faster in the beginning, as is the case with deep and slim trees like those of bell5, misc07, and mod008, the waistline will be much less in the real tree than in the measurement tree. This causes significant overestimations by the linear model.

One approach to better model the behavior of γ observed in the slim trees is to use a nonlinear model of the γ-sequence. For example, one could use a convex combination of the linear model γ and its cubic perturbation:

$$\tilde{\gamma}_i = \lambda(\bar{\gamma}_i - 1)^3 + (1-\lambda)(\bar{\gamma}_i - 1) + 1, \qquad \text{for } 0 \le i \le d_T$$

where $\lambda \in [0,1]$ and $\bar{\gamma}_i$ is the γ-sequence obtained by the linear model. This model has the linear model as a special case, when $\lambda = 0$. Increasing λ, it can be tuned to output good estimations to γ-sequences of slim trees, but it does not provide good results for the most common tree profiles when λ is far from 0. The linear model performs better for general MIP problems but if we deal with a special class of problems, it might be worth analyzing the tree profile and tuning the model.

To illustrate the above, we performed tests with the 30 multidimensional knapsack problem instances of the group mknapcb1 (100 variables, 5 constraints) from Beasley 1990b, Chu 1998. As we observed in Section 3.3, the branch-and-bound trees of this type of problems are usually slim and the linear model overestimates their size. We tried the cubic model with $\lambda = 0.5$. For this choice of λ, in 28 of the 30 cases, the prediction by the cubic model is closer to the actual number of nodes than that of the linear model. The mean error factor of the estimations by the linear model is 1.84, while that of the cubic model estimation is 1.28. (Value of 1 means exact estimation.)

The cubic model is only one example of improvement. Different models based on different sets of parameters can also prove useful.

3.5.2. Estimating the waist

Another concern is the quality of the estimation of the tree parameters. Tests of the sensitivity of the linear model to changes in the parameters show that the waist is the most important one. Even small changes in it cause large variations in the number of nodes in the measurement tree, while the variations caused by changes in the depth and the last full level are less significant. On the other hand, our experiments show that d_t and l_t of a small (with respect to the complete tree T) partial tree t are better estimates of d_T and l_T, respectively, than b_t is an estimate of b_T.

State-of-the-art branch-and-bound algorithms employ node selection rules that are a combination of depth-first search and best bound search. For example, the default branch-and-bound algorithm of CPLEX 8.0 dives along a path until a node gets pruned and then continues from a best bound node. As a consequence, shortly after the start of the algorithm, the top levels are well studied and the depth is estimated with a good precision. The sampling error present in the partial tree affects mainly the determination of the waist.

The variability of b_t is shown in Figure 2. The left figure depicts the tree profiles after 320, 640, 1280, 2560, 5120, 10715 seconds of solution time of the problem called rout. It can be seen that the waist gradually increases with time up to its final value of 33. This is shown also in the figure on the right, where the thin horizontal line is the waist of the complete tree, b_T, and the thick solid line represents the waist of the partial trees as a function of solution time. As time elapses, the waist approaches that of the complete tree. Greater fluctuations are typical at the beginning of the solution procedure. Tests show that usually $b_t < b_T$ for a small partial tree t, and sometimes the difference is significant. Therefore, often b_t is not a good estimate of b_T.

In some cases, the error in waist estimation can be reduced by considering the levels with large width and taking the estimate of the waist to be the midpoint of these levels. We call this

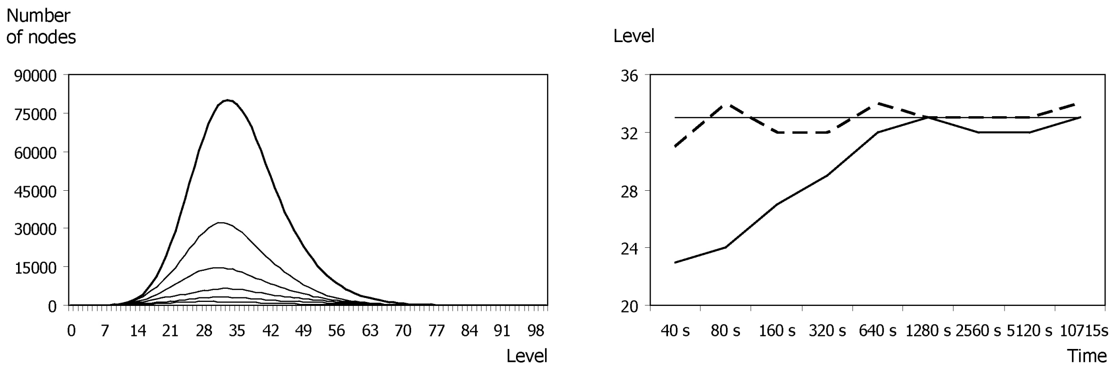

Figure 2: Evolution of the profile, the waist, and the average waist. Plotted profiles after 320, 640, 1280, 2560, 5120, 10715 seconds of solution time. Problem: rout.

the *average waist* and define it as follows:

Definition 3 *The* average waist *of a tree T is the level $\bar{b}_T = \left\lceil \frac{b_1+b_2}{2} \right\rceil$, where $b_1 = \min\{i : w_T(i) \geq 0.5t\}$, $b_2 = \max\{i : w_T(i) \geq 0.5t\}$, and $t = \max\{w_T(i) : 0 \leq i \leq d_T\}$, i.e., $\bar{b}_T$ is the center of the smallest interval containing the levels with width at least 50% of the maximum width.*

It is not uncommon that $\bar{b}_T \neq b_T$ for the complete tree T, but our tests indicate that both values are close. Early on in the solution process, the average waist is often a better estimate of the waist of the complete tree. Additionally, compared to b_t, the average waist $\bar{b}_t$ shows less variation during the solution process. Therefore, the average waist can be used to improve the prediction when there is a large variation in the waist. The average waist of the partial tree of problem rout is plotted with a dash line in the right graph of Figure 2.

Repeating the experiments on the same 99 test instances by using the average waist instead of the waist, the number of correct predictions increased from 78 to 85. This is a reduction of the incorrect predictions by one third. Further research in this direction would be worthwhile.

3.5.3. Effect of the bound from the best feasible solution found

If a good upper bound (for a minimization problem) is not found early in the solution process, the method can produce very poor and erratic estimates. Even after a good upper bound is found, the method could still produce poor estimates if it is biased by early "deep dives." Depending on the diving strategy, the first few dives into the tree can be very deep compared to what they would have been with a good a priori upper bound. This biases the estimate of the depth. One simple idea to avoid this problem is to eliminate (post facto) any node whose lower bound exceeds the current upper bound, even though such nodes are technically part of the search tree. This method will produce the same estimate that would have been produced had the bound been known a priori and should reduce the bias.

3.5.4. Experiments with branch and cut

We repeated the experiment from Section 3.3 with the default settings of the CPLEX branch-and-cut algorithm. The results are reported in Table 6.

Table 6: Predictions by our estimation procedure for a branch-and-cut algorithm.

Problem	Predicted number of nodes	Actual number of nodes	Ratio	Phase I time [s]	Predicted time range	Actual solution time	
air05	1576	720	2.2	169	1 m – 35 m	2.8 m	
arki001	396870541	347635	1100	74	> 10 h	3.2 h	†
bell3a	4818	19001	0.25	5	5 s – 35 s	29.6 s	
bell4	11403	19599	0.58	5	5 s – 1 m	20.6 s	
bell5	302597	582887	0.52	5	1 m – 25 m	9.6 m	
blend2	1434224	3973	360	8	20 m – 7 h	13 s	†
gesa2_o	1511	670	2.3	5	5 s – 1 m	6 s	
harp2 *	168082	320884	0.52	41	5 m – 3 h	> 54 m	†
markshare1 *	54252828	54563771	1	5	> 2 h	> 10 h	
markshare2 *	789981661	41189904	19	5	> 10 h	> 8.4 h	
mas74	56129	6518567	0.01	5	20 s – 10 m	3.1 h	†
mas76	35890	559528	0.06	5	10 s – 4 m	10.5 m	†
misc07	1.07e+11	79952	1.3e+6	17	> 10 h	9.5 m	†
mod011	4190	9635	0.43	985	15 m – 6 h	2.1 h	
noswot	83316	8308673	0.01	5	30 s – 10 m	4.8 h	†
pk1	9251	540710	0.02	5	5 s – 2 m	18.1 m	†
pp08aCUTS	1715	1910	0.9	6	5 s – 1 m	14 s	
qiu	3355	10098	0.33	118	2 m – 45 m	14.5 m	
rout	93362	99119	0.94	34	5 m – 2 h	42.8 m	
seymour *	119836809	57420	2087	2504	> 10 h	> 10 h	
stein45	26468	60717	0.44	5	20 s – 5 m	2 m	
vpm2	2418	4328	0.56	5	5 s – 30 s	10 s	

* Instance not solved to completion.

† Incorrect time prediction.

Six instances, lseu, mod008, modglob, pp08a, rgn, and stein27, were solved during Phase I. They are not present in the table. Eighteen of the remaining instances were solved to optimality. For eleven of them the prediction is correct. There are six cases, arki001, blend2, mas74, misc07, pk1, and pp08aCUTS, with considerable error. The solution of four instances was interrupted after exceeding the time or space limit, and this was predicted correctly for three of them.

Compared with Table 3, there is a deterioration in the prediction in some cases but there are some improvements too (like with problem rout). Overall, the quality of the estimate is not significantly different.

4. Conclusion

We showed empirically that the branch-and-bound solution time of an MIP solver can be roughly estimated in the early stages of the solution process. We proposed a procedure for this estimation based on parameters of a small subtree. Our experiments showed that in a relatively short time, we can obtain sufficient information to predict the total running time with an error within a factor of five. This procedure can easily be built into an MIP solver. It is fast and does not interfere with the branch-and-bound algorithm.

It might be worth exploring γ-sequence models that are contingent on particular types of integer programming problems. One might also be able to obtain relevant information on the whole tree profile using other parameters of the tree. Our attempts to use the amount of pruning in the subtree were fruitless but more research in this direction would be interesting.

Acknowledgements

We thank Jeff Linderoth, Michel Van Caneghem and the referees for their helpful comments. This research was supported by NSF grant DMI-0098427 and ONR grant N00014-97-1-0196.

References

Anstreicher, K., N. Brixius, J-P. Goux, J. Linderoth. 2002. Solving large quadratic assignment problems on computational grids. *Mathematical Programming B* **91** 563–588.

Beasley, J. E. 1988. An algorithm for solving large capacitated warehouse location problems. *European Journal of Operational Research* **33** 314–325.

Beasley, J. E. 1990a. A lagrangian heuristic for set-covering problems, *Naval Research Logistics* **37** 151–164.

Beasley, J. E. 1990b. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* **41(11)** 1069–1072.

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

Brüngger, A., A. Marzetta, J. Clausen and M. Perregaard. 1998. Solving large-scale QAP problems in parallel with the search library ZRAM. *Journal of Parallel and Distributed Computing* **50** 157–169.

Chen, P. G. 1992. Heuristic sampling: a method for predicting the performance of tree search programs. *SIAM Journal on Computing* **21** 295–315.

Chu, P. C., J. E. Beasley. 1998. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics* **4** 63–86.

E.Falkenauer, E. 1994. A Hybrid Grouping Genetic Algorithm for Bin Packing. Working paper CRIF Industrial Management and Automation, CP 106 - P4, 50 av. F.D.Roosevelt, B-1050 Brussels, Belgium.

Knuth, D. E. 1975. Estimating the efficiency of backtracking programs. *Mathematics of Computing* **29** 121–136.

Linderoth, J., M. W. P. Savelsbergh. 1997. A Computational Study of Search Strategies for Mixed Integer Programming. Report LEC-97-12, Georgia Institute of Technology, Atlanta, GA.

Lobjois, L., M. Lemaitre. 1998. Branch-and-bound algorithm selection by performance prediction American Association for Artificial Intelligence, Menlo Park, CA.

Nemhauser, G. L., L. A. Wolsey. 1988. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, NY.

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

Wolsey, L. A. 1998. *Integer Programming* John Wiley and Sons, New York, NY.

Web site of Argonne National Laboratory, Mathematics and Computer Science Division. URL: ftp://ftp.mcs.anl.gov/neos/mip-bench/.

H. Mittelmann's web site. URL: ftp://plato.la.asu.edu/pub/milp/.