A Branch-and-Price Algorithm for the Generalized Assignment Problem

Martin Savelsbergh
Georgia Institute of Technology
School of Industrial and Systems Engineering
Atlanta, GA 30332-0205
USA

Abstract

The generalized assignment problem examines the maximum profit assignment of jobs to agents such that each job is assigned to precisely one agent subject to capacity restrictions on the agents. A new algorithm for the generalized assignment problem is presented that employs both column generation and branch-and-bound to obtain optimal integer solutions to a set partitioning formulation of the problem.

July 1993
Revised November 1994
Revised October 1995

1 Introduction

The Generalized Assignment Problem (GAP) examines the maximum profit assignment of $n$ jobs to $m$ agents such that each job is assigned to precisely one agent subject to capacity restrictions on the agents. Although interesting and useful in its own right, its main importance stems from the fact that it appears as a substructure in many models developed to solve real-world problems in areas such as vehicle routing, plant location, resource scheduling, and flexible manufacturing systems.

The GAP is easily shown to be NP-hard and a considerable body of literature exists on the search for effective enumeration algorithms to solve problems of a reasonable size to optimality [Ross and Soland 1975, Martello and Toth 1981, Fisher, Jaikumar, and Van Wassenhove 1986, Guignard and Rosenwein 1989, Karabakal, Bean, and Lohmann 1992]. A recent survey by Cattrysse and Van Wassenhove [1992] provides a comprehensive treatment of most of these methods.

In this paper, we present an algorithm for the GAP that employs both column generation and branch-and-bound to obtain optimal integer solutions to a set partitioning formulation of the problem. We discuss various branching strategies that allow column generation at any
node in the branch-and-bound tree. Therefore, the algorithm can be viewed as a branch-and-price algorithm that is similar in spirit to the branch-and-cut algorithms that allow row generation at any node of the branch-and-bound tree.

Many variations of the basic algorithm have been implemented using MINTO, a Mixed INTeger Optimizer [Nemhauser, Savelsbergh, and Sigismondi 1994]. MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class.

This paper is organized as follows. Section 2 introduces both the standard and the set partitioning based formulation for the GAP. Section 3 presents the basic branch-and-price algorithm and discusses issues related to column generation and branch-and-bound. Section 4 examines the various branching strategies. Section 5 covers various implementation issues and Section 6 describes the computational experiments that have been conducted. Finally, Section 7 examines approximation algorithms derived from the branch-and-price algorithm.

2 Formulations

In the GAP the objective is to find a maximum profit assignment of $n$ jobs to $m$ agents such that each job is assigned to precisely one agent subject to capacity restrictions on the agents. The standard integer programming formulation is the following

$$\max \sum_{1 \leq i \leq m, 1 \leq j \leq n} p_{ij} x_{ij}$$

subject to

$$\sum_{1 \leq j \leq n} x_{ij} = 1 \quad j \in \{1, \ldots, n\},$$

$$\sum_{1 \leq j \leq n} w_{ij} x_{ij} \leq c_i \quad i \in \{1, \ldots, m\},$$

$$x_{ij} \in \{0, 1\} \quad i \in \{1, \ldots, m\}, \ j \in \{1, \ldots, n\},$$

where $p_{ij} \in ZZ_+$ is the profit associated with assigning job $j$ to agent $i$, $w_{ij} \in ZZ_+$ the claim on the capacity of agent $i$ by job $j$ if it is assigned to agent $i$, $c_i \in ZZ_+$ the capacity of agent $i$, and $x_{ij}$ a 0-1 variable indicating whether job $j$ is assigned to agent $i$ ($x_{ij} = 1$) or not ($x_{ij} = 0$).
The formulation underlying the branch-and-price algorithm discussed in this has an exponential number of variables and can be viewed as a disaggregated version of the above formulation.

Let \( K = \{ x_1, x_2, \ldots, x_k \} \) be the set of all possible feasible assignments of jobs to agent \( i \), i.e., \( x_k = (x_{1k}, x_{2k}, \ldots, x_{nk}) \) is a feasible solution to

\[
\sum_{1 \leq j \leq n} w_{ij} x_{ijk} \leq c_i
\]

\( x_{ijk} \in \{0, 1\} \quad j \in \{1, \ldots, n\} \).

Let \( y_k^i \) for \( i \in \{1, \ldots, m\} \) and \( k \in K \) be a binary variable indicating whether a feasible assignment \( x_{ijk} \) is selected for agent \( i \) (\( y_k^i = 1 \)) or not (\( y_k^i = 0 \)). The GAP can now be formulated as

\[
\max \sum_{1 \leq i \leq m, 1 \leq k \leq k_i} \left( \sum_{1 \leq j \leq n} p_{ij} x_{ijk} \right) y_k^i
\]

subject to

\[
\sum_{1 \leq i \leq m, 1 \leq k \leq k_i} x_{ijk} y_k^i = 1 \quad j \in \{1, \ldots, n\},
\]

\[
\sum_{1 \leq k \leq k_i} y_k^i \leq 1 \quad i \in \{1, \ldots, m\},
\]

\[
y_k^i \in \{0, 1\} \quad i \in \{1, \ldots, m\}, k \in K_i,
\]

where the first set of constraints enforces that each job is assigned to precisely one agent and the second set of constraints enforces that at most one feasible assignment is selected for each agent. This set partitioning formulation has been used by Cattrysse, Salomon, and Van Wassenhove [1994] to develop an approximation algorithm for the GAP.

The 0-1 knapsack problem associated with agent \( i \) in the standard formulation, i.e.,

\[
\max \sum_{1 \leq j \leq n} p_{ij} x_{ij}
\]

subject to

\[
\sum_{1 \leq j \leq n} w_{ij} x_{ij} \leq c_i,
\]

\( x_{ij} \in \{0, 1\} \quad j \in \{1, \ldots, n\} \),
has been replaced in the disaggregated formulation, by

$$\max \sum_{1 \leq k \leq k_i} (\sum_{1 \leq j \leq n} p_{ij}x_{jk})y_k$$

subject to

$$\sum_{1 \leq k \leq k_i} y_k \leq 1,$$

where $x^i_1, ..., x^i_{k_i}$ are the integral solutions to the knapsack problem. Because the linear programming relaxation of a 0-1 knapsack problem contains the convex hull of the integer solutions, the LP relaxation of the disaggregated formulation provides a bound that is at least as tight as the bound provided by the LP relaxation of the standard formulation.

Observe that the disaggregated formulation is essentially obtained by applying Dantzig-Wolfe decomposition to the standard formulation, where the knapsack constraints have been placed in the subproblem. Consequently, the value of the bound provided by the LP relaxation of the disaggregated formulation is equal to the value of the Lagrangean dual obtained by dualizing the semi-assignment constraints, i.e.,

$$\min_{\lambda} \max_{1 \leq i \leq m, 1 \leq j \leq n} \sum_{1 \leq i \leq m, 1 \leq j \leq n} p_{ij}x_{ij} + \sum_{1 \leq j \leq n} \lambda_j (1 - \sum_{1 \leq i \leq m} x_{ij})$$

subject to

$$\sum_{1 \leq j \leq n} w_{ij}x_{ij} \leq c_i \quad j \in \{1, ..., n\},$$

$$x_{ij} \in \{0, 1\} \quad i \in \{1, ..., m\}, j \in \{1, ..., n\},$$

See, for example, Nemhauser and Wolsey [1988, Section II.3.6] for an exposition of the relation between Lagrangean relaxation and Dantzig-Wolfe decomposition. The algorithms of Fisher, Jaikumar, and Van Wassenhove [1986], Guignard and Rosenwein [1989], and Karabakal, Bean, and Lohmann [1992] are based on bounds obtained by solving the above Lagrangean dual.

Our computational experiments will show that the branch-and-price algorithm discussed in this paper, although in theory using the same bounds, outperforms the optimization algorithms of Fisher, Jaikumar and Van Wassenhove [1986], Guignard and Rosenwein [1989], and Karabakal, Bean, and Lohmann [1992]. A plausible explanation for this phenomenon is the fact that the use of the simplex method provides much better convergence properties than the use of subgradient and dual ascent methods for the solution of the Lagrangean dual.
Let $y$ be any feasible solution to the LP-relaxation of the disaggregated formulation and let $z_{ij} = \sum_{1 \leq k \leq k_i} x_{jk}^i y_k^i$, then $z$ constitutes a feasible solution to the LP-relaxation of the standard formulation. Furthermore, we have the following

**Proposition.** If $y_k^i$ is fractional, then there must be a $j$ such that $z_{ij}$ is fractional.

**Proof.** Suppose there is no job $j$ such that $z_{ij}$ is fractional. Let $F = \{k \in K_i \mid 0 < y_k^i < 1\}$ be the set of fractional variables associated with agent $i$. We may assume that $|F| \geq 2$, because if $F = \{p\}$, then $z_{ij} = \sum_{1 \leq k \leq k_i} x_{jk}^i y_k^i$ is fractional for every $j$ with $x_{jp}^i = 1$. Note that the convexity constraint associated with agent $i$ implies that $\sum_{k \in F} y_k^i \leq 1$. Therefore, $\sum_{k \in F} x_{jk}^i y_k^i \leq \sum_{k \in F} y_k^i \leq 1$ for $j = 1, \ldots, n$. Consequently, $\sum_{k \in F} x_{jk}^i y_k^i$ is either 0 or 1 for $j = 1, \ldots, n$. If $\sum_{k \in F} x_{jk}^i y_k^i = 0$, then $x_{jk}^i = 0$ for all $k \in F$; if $\sum_{k \in F} x_{jk}^i y_k^i = 1$, then $x_{jk}^i = 1$ for all $k \in F$. But that means that we have duplicate columns; a contradiction.

3 Branch-and-price algorithms

Column generation is a pricing scheme for solving large-scale linear programs (LPs). Instead of pricing out nonbasic variables by enumeration, in a column generation approach the most negative (or positive) reduced price is found by solving an optimization problem. Gilmore and Gomory [1961] introduced the column generation approach in the context of cutting stock problems. In their case, as in many other cases, the linear program is a relaxation of an integer program (IP). However, when an IP relaxation is solved by column generation, the solution is not necessarily integral and it is not clear how to obtain an optimal or even feasible integer solution to the IP since standard branch-and-bound techniques can interfere with the column generation algorithm. Recently, various researchers have started to develop customized branching strategies to handle these difficulties, e.g., Desrochers, Desrosiers, and Solomon [1992] for vehicle routing problems, Desrochers and Soumis [1989] and Anbil, Tanga, and Johnson [1991] for crew scheduling problems, and Vance, Barnhart, Johnson, and Nemhauser [1992] for cutting stock problems.

Consider the linear programming relaxation of the disaggregated formulation for the GAP. This master problem cannot be solved directly due to the exponential number of columns. However, a restricted master problem that considers only a subset of the columns can be solved directly using, for instance, the simplex method. Additional columns for the restricted master problem can be generated as needed by solving the pricing problem

$$\max_{1 \leq i \leq m} \{z(K P_i) - v_i\},$$

where $v_i$ is the optimal dual price from the solution to the restricted master problem associated with the convexity constraint of agent $i$ and $z(K P_i)$ is the value of the optimal
solution to the following knapsack problem

\[
\max \sum_{1 \leq j \leq n} (p_{i,j} - u_j) x^i_j
\]

subject to

\[
\sum_{1 \leq j \leq n} w_{i,j} x^i_j \leq c_i
\]

\[
x^i_j \in \{0, 1\} \quad j \in \{1, \ldots, n\}
\]

with \( u_j \) being the optimal dual price from the solution to the restricted master problem associated with the partitioning constraint of job \( j \). A column prices out favorably to enter the basis if its reduced cost is positive. Consequently, if the objective value of the column generation subproblem is less than or equal to zero, then the current optimal solution for the restricted master problem is also optimal for the (unrestricted) master problem.

However, unless the \( y^k_i \)'s are integer, the solution to the master problem is not a solution to the original IP. In fact, there may not even be a feasible integer solution among the columns present in the master problem. However, computational experiments have indicated that the value of the LP relaxation does provide a very tight bound on the value of the optimal IP solution.

Applying a standard branch-and-bound procedure to the master problem with its existing columns will not guarantee an optimal (or feasible) solution. After branching, it may be the case that there exists a feasible assignment that would price out favorably, but this assignment is not present in the master problem. Therefore, to find an optimal solution we must generate columns after branching. However, suppose that we use the conventional branching rule based on variable dichotomy, we branch on the fractional variable \( y^k_i \), and we are in the branch in which \( y^k_i \) is fixed to zero. In the column generation phase, it is possible (and quite likely) that the optimal solution to the subproblem will be the same assignment represented by \( y^k_i \). In that case, it becomes necessary to generate the column with the \( 2^{nd} \) highest reduced cost. At depth \( n \) in the branch-and-bound tree we may need to find the column with \( n^{th} \) highest reduced cost.

In order to prevent columns that have been branched on from being regenerated, we must choose a branching rule that is compatible with the pricing problem. By compatible, we mean that we must be able to modify the subproblem so that columns that are infeasible due to the branching constraints will not be generated and the column generation subproblem will remain tractable.
4 Branching strategies

The challenge in formulating a branching strategy is to find one that excludes the current solution, validly partitions the solution space of the problem, and provides a pricing problem that is still tractable.

We have indicated in Section 2 that any feasible solution to the disaggregated formulation has a corresponding feasible solution to the standard formulation, and that if a solution to the disaggregated formulation is fractional, then the corresponding solution to the standard formulation is also fractional.

The idea now is to perform branching using the standard formulation while working with the disaggregated formulation. Branching strategies for 0-1 linear programs are based on fixing variables, either a single variable (variable dichotomy) or a set of variables (GUB dichotomy). Therefore, for our idea to work, we have to show that fixing a single variable or fixing a set of variables in the standard formulation has an equivalent in the disaggregated formulation, and that the resulting branching scheme is compatible with the pricing problem.

In the standard formulation, fixing variable $x_{ij}$ to zero forbids job $j$ to be assigned to agent $i$ and fixing variable $x_{ij}$ to one requires job $j$ to be assigned to agent $i$. In the disaggregated formulation this can be accomplished as follows. To forbid a job $j$ to be assigned to agent $i$, all variables for columns associated with agent $i$ that have a one in the row corresponding to job $j$ are fixed to zero, i.e., if $x_{ijk}^i = 1$, then $y_{ik} = 0$ for all $k \in K_i$. To require a job $j$ to be assigned to agent $i$, all variables for columns associated with agent $i$ that do not have a one in the row corresponding to job $j$ are fixed to zero, i.e., if $x_{ijk}^i = 0$, then $y_{ik} = 0$ for all $k \in K_i$, and all variables for columns not associated with agent $i$ that have a one in the row corresponding to job $j$ are fixed to zero, i.e., if $x_{ljk}^l = 1$, then $y_{lk} = 0$ for $1 \leq l \neq i \leq m$ and $k \in K_l$.

It is not hard to see that the resulting branching scheme is also compatible with the pricing problem. The pricing problem involves the solution of a knapsack problem for each agent. Forbidding the assignment of job $j$ to agent $i$ is accomplished by not considering job $j$ in the knapsack for agent $i$, and requiring the assignment of job $j$ to agent $i$ is accomplished by not considering job $j$ in the knapsack for agent $i$ and reducing the capacity of knapsack $i$ by the claim on its capacity by job $j$.

5 Implementation issues

5.1 Initial restricted master problem

To start the column generation procedure, an initial restricted master problem has to be provided. This initial restricted master problem must have a feasible LP relaxation to
ensure that proper dual information is passed to the pricing problem. We have chosen to start with one column for each agent, corresponding to the optimal knapsack solution, and a dummy column consisting of all ones with a large negative profit. The dummy column ensures that a feasible solution to the LP relaxation exists. This dummy column will be kept at all nodes of the branch-and-bound tree for the same reason.

5.2 Column generation subproblem

Any column with positive reduced cost is a candidate to enter the basis. The pricing problem defined above finds the column with highest reduced cost. Therefore, if a column with positive reduced cost exists the column generation will always identify a candidate column. This guarantees that the optimal solution to the linear program will be found.

However, solving the column generation problem involves the solution of several knapsack problems, which may be computationally prohibitive. Fortunately, for the column generation scheme to work, it is not necessary to always select the column with the highest reduced cost; any column with a positive reduced cost will do.

Therefore, various alternative column generation schemes can be developed. An obvious alternative is to select the first column encountered with a positive reduced cost. (To prevent a bias towards a certain agent a random starting point can be used.) This reduces the computation time per iteration. However, since the number of iterations may increase it is not sure whether the overall effect is positive. Another alternative is to select all columns encountered with a positive reduced cost. It is hard to estimate the effect on the computation time. Obviously, it does not affect the time required to solve the pricing problem, but it probably increases the time required to solve the restricted master, and it may increase or decrease the number of iterations.

Yet another alternative is the use of approximation algorithms to solve the pricing problem. To guarantee that the optimal solution to the linear program will be found a two-phase approach has to be used. A fast approximation algorithm is used to solve the pricing problem as long as it is able to identify a column with positive reduced cost. In case the approximation algorithm fails to identify a column with positive reduced cost, an optimization algorithm is invoked to prove optimality or generate a column with positive reduced cost. This process is repeated until the linear program is solved to optimality. We have not explored this alternative, since the pricing problem that has to be solved is a fairly small knapsack problem and there exist very efficient optimization algorithms to solve such problems.
5.3 Primal heuristics

It is well-known that the availability of good feasible solutions may reduce the size of the branch-and-bound tree considerably. The approximation algorithm that has been incorporated in our branch-and-price algorithm is a combination of the algorithms proposed by Martello and Toth [1981] and Jörnsten and Nasberg [1986].

Martello and Toth [1981] developed the following two-phase approximation algorithm for the GAP. Let \( f_{ij} \) be a measure of the ‘desirability’ of assigning job \( j \) to agent \( i \). (Martello and Toth suggest four measures: \( f_{ij} = p_{ij} \), \( f_{ij} = p_{ij}/w_{ij} \), \( f_{ij} = -w_{ij} \), and \( f_{ij} = -w_{ij}/c_{ij} \).) In the first phase, an attempt is made to construct an initial feasible solution. Iteratively consider all unassigned jobs, and determine the job \( j^* \) having maximum difference between the largest and the second largest \( f_{ij} \) for \( 1 \leq i \leq m \); job \( j^* \) is then assigned to agent for which \( f_{ij^*} \) is maximum. In the second phase, if a feasible solution has been found, the solution is improved through local exchanges.

The approximation algorithm developed by Jörnsten and Nasberg [1986] relies heavily on local exchanges. An initial solution is constructed by assigning each job to its most profitable agent. If this solution is feasible, which rarely happens, it is also optimal. Otherwise, some of the capacity constraints are violated. Next, local exchanges, using some infeasibility measure, are applied to obtain a feasible solution. Finally, if a feasible solution has been found, local exchanges are applied again, but now to improve the quality of the solution.

The approximation algorithm incorporated in our branch-and-price algorithm is basically the Martello and Toth algorithm extended with local exchange procedures to handle the situation in which the attempt to construct an initial feasible solution failed. It is invoked at every node of the branch-and-bound tree using \( f_{ij} = z_{ij} \), i.e., \( f_{ij} = \sum_{1 \leq k \leq \ell_i} x^i_{jk} y^i_k \), as a measure of the ‘desirability’ of assigning job \( j \) to agent \( i \).

5.4 Branching scheme

In Section 4, we have shown that branching strategies based on variable fixing are compatible with the pricing problem. We have explored two such branching strategies. In the first, we branch on the fractional variable \( x_{ij} \) with fractional part closest to 0.5. We set \( x_{ij} = 1 \) on one branch and \( x_{ij} = 0 \) on the other branch. In case of ties, we select the variable with highest profit \( p_{ij} \). In the second, we branch on the subset-sum constraint \( \sum_{1 \leq i \leq \ell_i} x_{ij} \) that contains the most fractional variables. We set \( \sum_{1 \leq i \leq \ell_i} x_{ij} = 0 \) on one branch and \( \sum_{\ell_i < i \leq m} x_{ij} = 0 \) on the other branch, where we have chosen \( \ell_i \) to be as close as possible to \( m/2 \); note that there has to be at least one fractional variable on both sides of \( \ell_i \). In case of ties, we select the subset-sum constraint for which \( |\sum_{1 \leq i \leq \ell_i} x_{ij} - 0.5| + |\sum_{\ell_i < i \leq m} x_{ij} - 0.5| \) is as small as possible.

The above branching strategies specify how the current set of feasible solutions is to be
divided into two smaller subsets. They do not specify how the subproblem to be solved next is to be selected. We have considered two selection strategies: depth-first search and best-bound search. Depth-first search is usually applied to get (hopefully good) feasible solutions fast; experience shows that feasible solutions are more likely to be found deep in the tree than at nodes near the root. Having a good feasible solution is necessary to be able to prune nodes and thus to reduce the size of the branch-and-bound tree. Best-bound search is motivated by the observation that the node containing the best bound has to be considered to prove optimality, so it may as well be explored first.

Recall that in our branch-and-price algorithm for the GAP, we invoke our primal heuristic at each node of the branch-and-bound tree and that the primal heuristic uses the current LP solution to measure desirability of assignments. Furthermore, observe that a best-bound search strategy jumps around the branch-and-bound tree much more than a depth-first search strategy. Consequently, if a best-bound search strategy is used, the primal heuristic is more likely to see different measures of desirability of assignments early on in the search process than when a depth-first search strategy is used. This increases the chance that the primal heuristic will identify a good feasible solution.

6 Computation results

The branch-and-price algorithm has been implemented using MINTO, a Mixed INTeger Optimizer [Nemhauser, Savelsbergh, and Sigismondi 1994]. MINTO is a software system that solves mixed-integer linear programs by a branch-and-bound algorithm with linear programming relaxations. It also provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. Moreover, the user can enrich the basic algorithm by providing a variety of specialized application routines that can customize MINTO to achieve maximum efficiency for a problem class. MINTO can either be built on top of the CPLEX callable library or on top of IBM’s Optimization Subroutine Library (OSL). Unless stated otherwise, our computational experiments have been conducted with MINTO 2.0/CPLEX 3.0 and have been run on an IBM/RS6000 model 590. The algorithm of Horowitz and Sahni [1974] has been used to solve the 0-1 knapsack problems.

We have conducted four computational experiments to determine the effectiveness and efficiency of our branch-and-price algorithm.

Optimization algorithms for the GAP are generally tested on four classes of random problems, usually referred to as A, B, C, and D, generated according to the following rules (see for instance Guignard and Rosenwein [1989]):

A. $p_{ij}$ and $w_{ij}$ are integer from a uniform distribution between 10 and 25 and between 5 and 25, respectively. $c_i = 9(n/m) + 0.4\max_{1 \leq i \leq m, \sum_{j \in J_i} w_{ij}} w_{ij}$, where $J_i^* = \{j | i = \arg\min_{1 \leq r \leq n} p_{rj}\}$. 

10
B. Same as A for $p_{ij}$ and $w_{ij}$. $c_i = 0.7$ of $c_i$ in A.

C. Same as A for $p_{ij}$ and $w_{ij}$. $c_i = 0.8 \sum_{1 \leq j \leq n} w_{ij} / m$.

D. Same as C for $c$. $w_{ij}$ is integer from uniform distribution between 1 and 100. $p_{ij} = 100 - w_{ij} + k$, where $k$ is integer from a uniform distribution between 1 and 21.

The above scheme generates instances for the minimization form of the GAP. Since our algorithm is designed for the maximization form of the GAP. All instances are converted to the maximization form by the following transformation. Let $t = \max_{1 \leq i \leq m, 1 \leq j \leq n} p_{ij} + 1$. We replace $p_{ij}$ by $t - p_{ij}$ for all $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$.

In the first two experiments, we have concentrated on identifying the best choices for the pricing scheme and the branching scheme. For these experiments we have used various sizes ($m = 5, 10, 20$ and $n = 30, 40$) for all problem classes (A, B, C, and D). All computational results presented are based on 10 randomly generated instances for the size and class under consideration.

In the first experiment, we have concentrated on the influence of the chosen algorithm for the pricing problem. For the set of instances, we have compared the all-positive, the best-positive, and the first-positive strategies. The results can be found in Table 1. The computational results show that all strategies have a comparable performance. For the next experiments, we have chosen to use the all-positive strategy, since this strategy required the smallest amount of computation time over all instances.

In the second experiment, we have concentrated on the influence of the chosen branching strategy. For the set of instances, we have compared the division schemes based on variable dichotomy and on GUB dichotomy, and the depth-first and best-bound selection schemes. The results can be found in Tables 2 and 3. The computational results show that the best-bound selection scheme clearly outperforms the depth-first selection scheme, and that the division scheme based on GUB dichotomy and the division scheme based on variable dichotomy have a comparable performance. For the next experiments, we have chosen to use the best-bound selection scheme and the division scheme based on GUB dichotomy, since this strategy required the smallest amount of computation time over all instances.

In the final two experiments, we have concentrated on the overall performance of our branch-and-price algorithm. For these experiments we have used various sizes ($m = 3, 5, 10, 20$ and $n = 30, 50$) for all problem classes (A, B, C, and D). All computational results presented are based on 10 randomly generated instances for the size and class under consideration. Note that many instances in this test set are larger than those that have been used in computational experiments reported in earlier papers.

In the third experiment, we have concentrated on the quality of the bounds. For the test instances, we have compared the value of the linear programming relaxation of the standard formulation ($LP_1$), the value of the linear programming relaxation of the standard
The lifted knapsack covers are automatically generated by MINTO. For a description of the specific algorithms embedded in MINTO we refer the reader to Gu, Nemhauser, and Savelsbergh [1994]. The results can be found in Tables 4 and 5. The computational results in Table 4 show that the linear programming bound of the disaggregated formulation is quite good. Even for the instances in the most difficult problem class D, the average integrality gap is less than 0.5 percent. A closer examination also reveals that the difference in quality between $LP_1$ and $LP_3$ is larger for instances with a small ratio $\frac{n}{m}$. This phenomenon can be explained as follows. The ratio $\frac{n}{m}$ represents the average number of jobs assigned to agents. If the average number of jobs assigned to agents is small, then the LP relaxations of the 0-1 knapsack problems associated with the agents are typically weak. Consequently,
Table 2: Comparison of branching division schemes for best-bound selection

<table>
<thead>
<tr>
<th>Instance</th>
<th>GUB-dichotomy</th>
<th>variable-dichotomy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#nodes</td>
<td>cpu</td>
</tr>
<tr>
<td>A.5.30</td>
<td>avg</td>
<td>max</td>
</tr>
<tr>
<td>A.10.30</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>A.20.30</td>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>A.5.40</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>A.10.40</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>A.20.40</td>
<td>1.6</td>
<td>4</td>
</tr>
<tr>
<td>B.5.30</td>
<td>1.2</td>
<td>3</td>
</tr>
<tr>
<td>B.10.30</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>B.20.30</td>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>B.5.40</td>
<td>6.2</td>
<td>15</td>
</tr>
<tr>
<td>B.10.40</td>
<td>2.5</td>
<td>7</td>
</tr>
<tr>
<td>B.20.40</td>
<td>2.9</td>
<td>6</td>
</tr>
<tr>
<td>C.5.30</td>
<td>2.7</td>
<td>8</td>
</tr>
<tr>
<td>C.10.30</td>
<td>4.1</td>
<td>23</td>
</tr>
<tr>
<td>C.20.30</td>
<td>1.3</td>
<td>3</td>
</tr>
<tr>
<td>C.5.40</td>
<td>7.9</td>
<td>19</td>
</tr>
<tr>
<td>C.10.40</td>
<td>5.4</td>
<td>17</td>
</tr>
<tr>
<td>C.20.40</td>
<td>2.9</td>
<td>13</td>
</tr>
<tr>
<td>D.5.30</td>
<td>44.2</td>
<td>191</td>
</tr>
<tr>
<td>D.10.30</td>
<td>77.3</td>
<td>199</td>
</tr>
<tr>
<td>D.20.30</td>
<td>6.0</td>
<td>20</td>
</tr>
<tr>
<td>D.5.40</td>
<td>53.3</td>
<td>200</td>
</tr>
<tr>
<td>D.10.40</td>
<td>187.6</td>
<td>749</td>
</tr>
<tr>
<td>D.20.40</td>
<td>70.3</td>
<td>213</td>
</tr>
</tbody>
</table>

Solving these 0-1 knapsacks to optimality, as done in the disaggregated formulation, will lead to substantially stronger bounds. If the average number of jobs assigned to agents is large, then the LP relaxations of the 0-1 knapsack problems associated with the agents are typically strong, and solving them to optimality will not lead to substantially stronger bounds.

The computational results in Table 5 show that the increased quality of the bounds comes at a price. The computation times have increased as well, especially on problem classes with a high ratio $\frac{n}{m}$. This phenomenon can be explained as follows. If the average number of jobs assigned to agents is small, then the 0-1 knapsack problems associated with the agents typically have a small number of feasible solutions, which is a favorable situation for a column generation approach. If the average number of jobs assigned to agents is large, then the 0-1 knapsack problems associated with the agents typically have a large number of feasible solutions, many of which may have comparable objective function values, which is not a favorable situation for column generation approaches.

The arguments presented above indicate that we can expect our branch-and-price algo-
Table 3: Comparison of division schemes for depth-first selection

<table>
<thead>
<tr>
<th>Instance</th>
<th>GUB-dichotomy</th>
<th>variable-dichotomy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#nodes</td>
<td>cpu</td>
</tr>
<tr>
<td></td>
<td>avg.</td>
<td>max.</td>
</tr>
<tr>
<td>A.5.30</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>A.10.30</td>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>A.20.30</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>A.5.40</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>A.10.40</td>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>A.20.40</td>
<td>1.6</td>
<td>4</td>
</tr>
<tr>
<td>B.5.30</td>
<td>1.2</td>
<td>3</td>
</tr>
<tr>
<td>B.10.30</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>B.20.30</td>
<td>1.5</td>
<td>4</td>
</tr>
<tr>
<td>B.5.40</td>
<td>7.3</td>
<td>26</td>
</tr>
<tr>
<td>B.10.40</td>
<td>3.0</td>
<td>8</td>
</tr>
<tr>
<td>B.20.40</td>
<td>2.5</td>
<td>5</td>
</tr>
<tr>
<td>C.5.30</td>
<td>3.1</td>
<td>11</td>
</tr>
<tr>
<td>C.10.30</td>
<td>5.3</td>
<td>27</td>
</tr>
<tr>
<td>C.20.30</td>
<td>1.3</td>
<td>3</td>
</tr>
<tr>
<td>C.10.40</td>
<td>5.1</td>
<td>14</td>
</tr>
<tr>
<td>C.20.40</td>
<td>2.8</td>
<td>13</td>
</tr>
<tr>
<td>D.5.30</td>
<td>140.0</td>
<td>455</td>
</tr>
<tr>
<td>D.10.30</td>
<td>3.7</td>
<td>779</td>
</tr>
<tr>
<td>D.20.30</td>
<td>32.9</td>
<td>82</td>
</tr>
<tr>
<td>D.5.40</td>
<td>239.0</td>
<td>585</td>
</tr>
<tr>
<td>D.10.40</td>
<td>1183.6</td>
<td>2036</td>
</tr>
<tr>
<td>D.20.40</td>
<td>575.9</td>
<td>1345</td>
</tr>
</tbody>
</table>

The computational results show that, as anticipated, our branch-and-price algorithm performs better on problem classes with a relatively small ratio $\frac{n}{m}$, i.e., $\frac{n}{m} \leq 5$, whereas the Karabakal, Bean, and Lohmann algorithm performs better on problem classes with a large ratio $\frac{n}{m}$, i.e., $\frac{n}{m} > 5$. As such, the branch-and-price algorithm and the Lagrangian dual algorithm complement each other nicely.
7 Approximation algorithms

The quality of the linear programming bound associated with the disaggregated formulation and the fact that good feasible solutions are usually found early on in the solution process suggest that truncated tree search algorithms may provide very good approximation algorithms. In truncated tree search algorithms the number of nodes evaluated in the solution process is reduced according to some prespecified scheme. Truncated tree search algorithms present a trade-off between effectiveness and efficiency.

We have considered two different schemes to reduce the number of evaluated nodes in the solution process.

In the first scheme, no more than $\mu$, a prespecified fixed number, nodes will be evaluated. The advantage of this scheme is the fact that it guarantees that the time required to produce a solution is fairly predictable. The disadvantage is that it is impossible to say anything beforehand about the quality of the solution produced by the algorithm (it is not even guaranteed that a solution will be found).

In the second scheme, a node is fathomed if $z_{LP} \leq (1 + \alpha) \cdot z_{IP}$, where $z_{LP}$ is the value of the linear programming solution at the node, $z_{IP}$ the value of the best known integer programming solution, and $\alpha > 0$ an optimality tolerance. The advantage of this scheme is the fact that it guarantees that the value of the solution produced by the algorithm is within $\alpha \cdot 100$ percent of the optimal value. The disadvantage is that it is impossible to say anything beforehand on the time required to produce a solution.

Very few approximation algorithms exist for the GAP. Most of them consist of two phases: a construction phase, in which an initial feasible solution is constructed, and an improvement phase, in which the initial feasible solution is improved. To the best of our knowledge the linear relaxation heuristic (LRH) proposed by Trick [1992] is one of the best among these heuristics in terms of quality of solution as well as solution times.

In an independent study, Cattrysse, Salomon, and Van Wassenhove [1994] have used the set partitioning formulation underlying the algorithms developed and discussed in this paper to develop an approximation algorithm for the GAP. Their algorithm consists of two phases. In the first phase, the linear programming relaxation of the master problem is solved approximately. In the process a set of columns, i.e., feasible assignments of jobs to machines, is obtained. In the second phase, an enumeration scheme developed by Garfinkel and Nemhauser is used to identify a feasible solution to the GAP among the columns generated in the first phase. They have tested their approximation algorithm on instances of various sizes from class C.

We have compared the performance of three truncated tree search algorithms, with $\mu = 10$, $\alpha = 0.01$ and $\alpha = 0.005$, to the performance of the linear relaxation heuristic on ten randomly generated instances in the problem classes $(D, 10, 50)$ and $(D, 20, 50)$, and on two sets of ten even larger randomly generated instances in the problem classes $(D, 10, 100)$
and \((D, 20, 100)\). The results can be found in Tables 8, 9, 10, and 11. These computational experiments have been conducted with MINTO 1.6/CPLEX 2.1 and have been run on an IBM/RS6000 model 550. The computational results show that the truncated tree search algorithms clearly outperform the linear relaxation heuristic in terms of solution quality with an acceptable increase in computation time for \(\mu = 10\) and \(\alpha = 0.01\) and a considerable increase in computation time for \(\alpha = 0.005\). All in all, the truncated tree search algorithms provide a good balance between effectiveness and efficiency.

**Acknowledgment**

We would like to thank James Bean and Mike Trick for making their code available to us. Furthermore, we would like to thank an anonymous referee for his comments and suggestions, which have helped improve the quality and readability of the paper.

**References**


Table 4: Quality of the bounds I: integrality gap

<table>
<thead>
<tr>
<th>Instance</th>
<th>n/m</th>
<th>( LP_1 )</th>
<th>( LP_2 )</th>
<th>( LP_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>avg.</td>
<td>max.</td>
<td>avg.</td>
</tr>
<tr>
<td>A.3:30</td>
<td>10.0</td>
<td>0.241</td>
<td>0.641</td>
<td>0.114</td>
</tr>
<tr>
<td>A.5:30</td>
<td>6.0</td>
<td>0.196</td>
<td>0.309</td>
<td>0.117</td>
</tr>
<tr>
<td>A.10:30</td>
<td>3.0</td>
<td>0.169</td>
<td>0.400</td>
<td>0.000</td>
</tr>
<tr>
<td>A.20:30</td>
<td>1.5</td>
<td>0.178</td>
<td>0.614</td>
<td>0.000</td>
</tr>
<tr>
<td>A.3:50</td>
<td>16.7</td>
<td>0.089</td>
<td>0.179</td>
<td>0.055</td>
</tr>
<tr>
<td>A.5:50</td>
<td>10.0</td>
<td>0.065</td>
<td>0.198</td>
<td>0.039</td>
</tr>
<tr>
<td>A.10:50</td>
<td>5.0</td>
<td>0.130</td>
<td>0.513</td>
<td>0.000</td>
</tr>
<tr>
<td>A.20:50</td>
<td>2.5</td>
<td>0.102</td>
<td>0.269</td>
<td>0.000</td>
</tr>
<tr>
<td>B.3:30</td>
<td>10.0</td>
<td>2.189</td>
<td>4.887</td>
<td>1.173</td>
</tr>
<tr>
<td>B.5:30</td>
<td>6.0</td>
<td>1.417</td>
<td>3.840</td>
<td>0.421</td>
</tr>
<tr>
<td>B.10:30</td>
<td>3.0</td>
<td>0.784</td>
<td>1.471</td>
<td>0.057</td>
</tr>
<tr>
<td>B.20:30</td>
<td>1.5</td>
<td>1.004</td>
<td>2.124</td>
<td>0.022</td>
</tr>
<tr>
<td>B.3:50</td>
<td>16.7</td>
<td>1.003</td>
<td>1.949</td>
<td>0.846</td>
</tr>
<tr>
<td>B.5:50</td>
<td>10.0</td>
<td>0.888</td>
<td>2.053</td>
<td>0.517</td>
</tr>
<tr>
<td>B.10:50</td>
<td>5.0</td>
<td>0.664</td>
<td>1.409</td>
<td>0.188</td>
</tr>
<tr>
<td>B.20:50</td>
<td>2.5</td>
<td>0.315</td>
<td>0.662</td>
<td>0.017</td>
</tr>
<tr>
<td>C.3:30</td>
<td>10.0</td>
<td>1.770</td>
<td>3.021</td>
<td>0.981</td>
</tr>
<tr>
<td>C.5:30</td>
<td>6.0</td>
<td>2.284</td>
<td>4.030</td>
<td>0.865</td>
</tr>
<tr>
<td>C.10:30</td>
<td>3.0</td>
<td>2.305</td>
<td>3.385</td>
<td>0.377</td>
</tr>
<tr>
<td>C.20:30</td>
<td>1.5</td>
<td>4.067</td>
<td>6.793</td>
<td>0.042</td>
</tr>
<tr>
<td>C.3:50</td>
<td>16.7</td>
<td>0.282</td>
<td>1.241</td>
<td>0.683</td>
</tr>
<tr>
<td>C.5:50</td>
<td>10.0</td>
<td>0.829</td>
<td>1.181</td>
<td>0.549</td>
</tr>
<tr>
<td>C.10:50</td>
<td>5.0</td>
<td>1.108</td>
<td>1.603</td>
<td>0.437</td>
</tr>
<tr>
<td>C.20:50</td>
<td>2.5</td>
<td>1.760</td>
<td>2.155</td>
<td>0.219</td>
</tr>
<tr>
<td>D.3:30</td>
<td>10.0</td>
<td>1.132</td>
<td>1.414</td>
<td>0.978</td>
</tr>
<tr>
<td>D.5:30</td>
<td>6.0</td>
<td>1.628</td>
<td>2.093</td>
<td>1.329</td>
</tr>
<tr>
<td>D.10:30</td>
<td>3.0</td>
<td>3.369</td>
<td>4.055</td>
<td>2.172</td>
</tr>
<tr>
<td>D.20:30</td>
<td>1.5</td>
<td>9.145</td>
<td>10.899</td>
<td>2.027</td>
</tr>
<tr>
<td>D.3:50</td>
<td>16.7</td>
<td>0.460</td>
<td>0.930</td>
<td>0.393</td>
</tr>
<tr>
<td>D.5:50</td>
<td>10.0</td>
<td>0.392</td>
<td>0.783</td>
<td>0.527</td>
</tr>
<tr>
<td>D.10:50</td>
<td>5.0</td>
<td>1.364</td>
<td>1.685</td>
<td>1.122</td>
</tr>
<tr>
<td>D.20:50</td>
<td>2.5</td>
<td>2.997</td>
<td>3.253</td>
<td>1.975</td>
</tr>
</tbody>
</table>
Table 5: Quality of the bounds II: computation time

<table>
<thead>
<tr>
<th>Instance</th>
<th>m/n</th>
<th>LP_1</th>
<th></th>
<th></th>
<th>LP_2</th>
<th></th>
<th></th>
<th>LP_3</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg.</td>
<td>max.</td>
<td>avg.</td>
<td>max.</td>
<td>avg.</td>
<td>max.</td>
<td>avg.</td>
<td>max.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.3.30</td>
<td>10.0</td>
<td>0.102</td>
<td>0.12</td>
<td>0.136</td>
<td>0.25</td>
<td>1.822</td>
<td>2.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.5.30</td>
<td>6.0</td>
<td>0.111</td>
<td>0.12</td>
<td>0.128</td>
<td>0.15</td>
<td>1.006</td>
<td>1.20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.10.30</td>
<td>3.0</td>
<td>0.163</td>
<td>0.17</td>
<td>0.182</td>
<td>0.20</td>
<td>0.617</td>
<td>0.70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.20.30</td>
<td>1.5</td>
<td>0.229</td>
<td>0.25</td>
<td>0.263</td>
<td>0.30</td>
<td>0.428</td>
<td>0.55</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.3.50</td>
<td>16.7</td>
<td>0.119</td>
<td>0.13</td>
<td>0.177</td>
<td>0.27</td>
<td>23.361</td>
<td>29.80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.5.50</td>
<td>10.0</td>
<td>0.142</td>
<td>0.16</td>
<td>0.185</td>
<td>0.35</td>
<td>7.171</td>
<td>8.01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.10.50</td>
<td>5.0</td>
<td>0.229</td>
<td>0.25</td>
<td>0.236</td>
<td>0.28</td>
<td>3.110</td>
<td>3.76</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A.20.50</td>
<td>2.5</td>
<td>0.325</td>
<td>0.35</td>
<td>0.357</td>
<td>0.39</td>
<td>1.830</td>
<td>2.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.3.30</td>
<td>10.0</td>
<td>0.102</td>
<td>0.11</td>
<td>0.194</td>
<td>0.36</td>
<td>1.509</td>
<td>1.78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.3.30</td>
<td>6.0</td>
<td>0.124</td>
<td>0.14</td>
<td>0.266</td>
<td>0.52</td>
<td>0.834</td>
<td>0.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.10.30</td>
<td>3.0</td>
<td>0.172</td>
<td>0.18</td>
<td>0.219</td>
<td>0.28</td>
<td>0.489</td>
<td>0.53</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.20.30</td>
<td>1.5</td>
<td>0.172</td>
<td>0.18</td>
<td>0.314</td>
<td>0.39</td>
<td>0.365</td>
<td>0.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.3.50</td>
<td>16.7</td>
<td>0.128</td>
<td>0.16</td>
<td>0.239</td>
<td>0.32</td>
<td>17.494</td>
<td>21.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.5.50</td>
<td>10.0</td>
<td>0.155</td>
<td>0.16</td>
<td>0.365</td>
<td>0.50</td>
<td>5.553</td>
<td>6.98</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.10.50</td>
<td>5.0</td>
<td>0.245</td>
<td>0.27</td>
<td>0.342</td>
<td>0.48</td>
<td>2.287</td>
<td>2.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B.20.50</td>
<td>2.5</td>
<td>0.341</td>
<td>0.38</td>
<td>0.446</td>
<td>0.54</td>
<td>1.406</td>
<td>1.68</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.3.30</td>
<td>10.0</td>
<td>0.115</td>
<td>0.14</td>
<td>0.195</td>
<td>0.30</td>
<td>1.532</td>
<td>1.80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.5.30</td>
<td>6.0</td>
<td>0.128</td>
<td>0.16</td>
<td>0.290</td>
<td>0.44</td>
<td>0.765</td>
<td>0.91</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.10.30</td>
<td>3.0</td>
<td>0.178</td>
<td>0.19</td>
<td>0.310</td>
<td>0.37</td>
<td>0.445</td>
<td>0.51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.20.30</td>
<td>1.5</td>
<td>0.279</td>
<td>0.30</td>
<td>0.488</td>
<td>0.56</td>
<td>0.205</td>
<td>0.33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.3.50</td>
<td>16.7</td>
<td>0.135</td>
<td>0.15</td>
<td>0.239</td>
<td>0.28</td>
<td>17.243</td>
<td>22.87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.5.50</td>
<td>10.0</td>
<td>0.151</td>
<td>0.17</td>
<td>0.377</td>
<td>0.47</td>
<td>5.612</td>
<td>6.72</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.10.50</td>
<td>5.0</td>
<td>0.245</td>
<td>0.26</td>
<td>0.385</td>
<td>0.44</td>
<td>2.097</td>
<td>2.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C.20.50</td>
<td>2.5</td>
<td>0.419</td>
<td>0.44</td>
<td>0.840</td>
<td>0.99</td>
<td>1.142</td>
<td>1.24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.3.30</td>
<td>10.0</td>
<td>0.116</td>
<td>0.14</td>
<td>0.180</td>
<td>0.20</td>
<td>1.379</td>
<td>1.66</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.5.30</td>
<td>6.0</td>
<td>0.131</td>
<td>0.15</td>
<td>0.283</td>
<td>0.33</td>
<td>0.834</td>
<td>0.94</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.10.30</td>
<td>3.0</td>
<td>0.207</td>
<td>0.22</td>
<td>0.340</td>
<td>0.37</td>
<td>0.653</td>
<td>0.77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.20.30</td>
<td>1.5</td>
<td>0.311</td>
<td>0.34</td>
<td>1.926</td>
<td>2.63</td>
<td>0.519</td>
<td>0.58</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.3.50</td>
<td>16.7</td>
<td>0.141</td>
<td>0.16</td>
<td>0.281</td>
<td>0.31</td>
<td>14.682</td>
<td>18.51</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.5.50</td>
<td>10.0</td>
<td>0.177</td>
<td>0.19</td>
<td>0.464</td>
<td>0.54</td>
<td>5.833</td>
<td>6.61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.10.50</td>
<td>5.0</td>
<td>0.301</td>
<td>0.31</td>
<td>0.413</td>
<td>0.44</td>
<td>3.120</td>
<td>3.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D.20.50</td>
<td>2.5</td>
<td>0.523</td>
<td>0.36</td>
<td>1.097</td>
<td>1.30</td>
<td>2.357</td>
<td>2.46</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6: Results for the Karabakal, Bean, and Lohmann algorithm

<table>
<thead>
<tr>
<th>Instance</th>
<th>n/m</th>
<th>#nodes</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>avg.</td>
<td>max.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>avg.</td>
<td>max.</td>
</tr>
<tr>
<td>A.3.30</td>
<td>10.0</td>
<td>1.5</td>
<td>15</td>
</tr>
<tr>
<td>A.5.30</td>
<td>6.0</td>
<td>21.8</td>
<td>84</td>
</tr>
<tr>
<td>A.10.30</td>
<td>3.0</td>
<td>49.2</td>
<td>168</td>
</tr>
<tr>
<td>A.20.30</td>
<td>1.5</td>
<td>154.2</td>
<td>556</td>
</tr>
<tr>
<td>A.3.50</td>
<td>16.7</td>
<td>2.7</td>
<td>26</td>
</tr>
<tr>
<td>A.5.50</td>
<td>10.0</td>
<td>23.0</td>
<td>83</td>
</tr>
<tr>
<td>A.10.50</td>
<td>5.0</td>
<td>110.5</td>
<td>333</td>
</tr>
<tr>
<td>A.20.50</td>
<td>2.5</td>
<td>403.9</td>
<td>979</td>
</tr>
<tr>
<td>B.3.30</td>
<td>10.0</td>
<td>1641.7</td>
<td>14240</td>
</tr>
<tr>
<td>B.5.30</td>
<td>6.0</td>
<td>54.1</td>
<td>115</td>
</tr>
<tr>
<td>B.10.30</td>
<td>3.0</td>
<td>69.2</td>
<td>234</td>
</tr>
<tr>
<td>B.20.30</td>
<td>1.5</td>
<td>351.3</td>
<td>770</td>
</tr>
<tr>
<td>B.3.50</td>
<td>16.7</td>
<td>1942.1</td>
<td>15653</td>
</tr>
<tr>
<td>B.5.50</td>
<td>10.0</td>
<td>241.2</td>
<td>777</td>
</tr>
<tr>
<td>B.10.50</td>
<td>5.0</td>
<td>308.6</td>
<td>1009</td>
</tr>
<tr>
<td>B.20.50</td>
<td>2.5</td>
<td>11789.0</td>
<td>84396</td>
</tr>
<tr>
<td>C.3.30</td>
<td>10.0</td>
<td>136.0</td>
<td>549</td>
</tr>
<tr>
<td>C.5.30</td>
<td>6.0</td>
<td>88.1</td>
<td>340</td>
</tr>
<tr>
<td>C.10.30</td>
<td>3.0</td>
<td>189.1</td>
<td>1212</td>
</tr>
<tr>
<td>C.20.30</td>
<td>1.5</td>
<td>281.1</td>
<td>665</td>
</tr>
<tr>
<td>C.3.50</td>
<td>16.7</td>
<td>313.6</td>
<td>558</td>
</tr>
<tr>
<td>C.5.50</td>
<td>10.0</td>
<td>243.2</td>
<td>896</td>
</tr>
<tr>
<td>C.10.50</td>
<td>5.0</td>
<td>1298.6</td>
<td>4910</td>
</tr>
<tr>
<td>C.20.50</td>
<td>2.5</td>
<td>8214.9</td>
<td>20123</td>
</tr>
<tr>
<td>D.3.30</td>
<td>10.0</td>
<td>179.4</td>
<td>525</td>
</tr>
<tr>
<td>D.5.30</td>
<td>6.0</td>
<td>498.2</td>
<td>2437</td>
</tr>
<tr>
<td>D.10.30</td>
<td>3.0</td>
<td>1793.8</td>
<td>3299</td>
</tr>
<tr>
<td>D.20.30</td>
<td>1.5</td>
<td>394.3</td>
<td>1034</td>
</tr>
<tr>
<td>D.3.50</td>
<td>16.7</td>
<td>531.6</td>
<td>1200</td>
</tr>
<tr>
<td>D.5.50</td>
<td>10.0</td>
<td>4182.1</td>
<td>13201</td>
</tr>
<tr>
<td>D.20.50</td>
<td>5.0</td>
<td>52136.5</td>
<td>96309</td>
</tr>
<tr>
<td>D.20.50 **</td>
<td>2.5</td>
<td>65206.5</td>
<td>91486</td>
</tr>
</tbody>
</table>

* Based on 8 instances solved within 100,000 nodes
** Based on 4 instances solved within 100,000 nodes
Table 7: Results for the Savelsbergh algorithm

<table>
<thead>
<tr>
<th>Instance</th>
<th>n/m</th>
<th>#nodes</th>
<th>cpu</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>avg.</td>
</tr>
<tr>
<td>A.3.30</td>
<td>10.0</td>
<td>2.0</td>
<td>5</td>
</tr>
<tr>
<td>A.3.30</td>
<td>6.0</td>
<td>2.0</td>
<td>5</td>
</tr>
<tr>
<td>A.10.30</td>
<td>3.0</td>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>A.20.30</td>
<td>1.5</td>
<td>1.2</td>
<td>2</td>
</tr>
<tr>
<td>A.3.50</td>
<td>16.7</td>
<td>2.5</td>
<td>5</td>
</tr>
<tr>
<td>A.5.50</td>
<td>10.0</td>
<td>2.0</td>
<td>5</td>
</tr>
<tr>
<td>A.10.50</td>
<td>5.0</td>
<td>1.9</td>
<td>6</td>
</tr>
<tr>
<td>A.20.50</td>
<td>2.5</td>
<td>1.6</td>
<td>3</td>
</tr>
<tr>
<td>B.3.30</td>
<td>10.0</td>
<td>3.9</td>
<td>12</td>
</tr>
<tr>
<td>B.3.30</td>
<td>6.0</td>
<td>2.3</td>
<td>5</td>
</tr>
<tr>
<td>B.10.30</td>
<td>3.0</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td>B.20.30</td>
<td>1.5</td>
<td>1.4</td>
<td>3</td>
</tr>
<tr>
<td>B.3.50</td>
<td>16.7</td>
<td>12.0</td>
<td>27</td>
</tr>
<tr>
<td>B.5.50</td>
<td>10.0</td>
<td>8.8</td>
<td>29</td>
</tr>
<tr>
<td>B.10.50</td>
<td>5.0</td>
<td>2.9</td>
<td>9</td>
</tr>
<tr>
<td>B.20.50</td>
<td>2.5</td>
<td>2.0</td>
<td>6</td>
</tr>
<tr>
<td>C.3.30</td>
<td>10.0</td>
<td>6.0</td>
<td>15</td>
</tr>
<tr>
<td>C.5.30</td>
<td>6.0</td>
<td>4.9</td>
<td>14</td>
</tr>
<tr>
<td>C.10.30</td>
<td>3.0</td>
<td>4.1</td>
<td>23</td>
</tr>
<tr>
<td>C.20.30</td>
<td>1.5</td>
<td>1.3</td>
<td>3</td>
</tr>
<tr>
<td>C.3.50</td>
<td>16.7</td>
<td>14.7</td>
<td>43</td>
</tr>
<tr>
<td>C.5.50</td>
<td>10.0</td>
<td>11.7</td>
<td>23</td>
</tr>
<tr>
<td>C.10.50</td>
<td>5.0</td>
<td>6.2</td>
<td>18</td>
</tr>
<tr>
<td>C.20.50</td>
<td>2.5</td>
<td>3.9</td>
<td>17</td>
</tr>
<tr>
<td>D.3.30</td>
<td>10.0</td>
<td>31.3</td>
<td>97</td>
</tr>
<tr>
<td>D.5.30</td>
<td>6.0</td>
<td>55.1</td>
<td>249</td>
</tr>
<tr>
<td>D.10.30</td>
<td>3.0</td>
<td>77.3</td>
<td>199</td>
</tr>
<tr>
<td>D.20.30</td>
<td>1.5</td>
<td>6.0</td>
<td>20</td>
</tr>
<tr>
<td>D.3.50</td>
<td>16.7</td>
<td>70.0</td>
<td>176</td>
</tr>
<tr>
<td>D.5.50</td>
<td>10.0</td>
<td>175.7</td>
<td>695</td>
</tr>
<tr>
<td>D.10.50</td>
<td>5.0</td>
<td>647.6</td>
<td>1896</td>
</tr>
<tr>
<td>D.20.50</td>
<td>2.5</td>
<td>395.0</td>
<td>1712</td>
</tr>
</tbody>
</table>

21
Table 8: Performance of truncated tree search algorithms (D,10,50)

<table>
<thead>
<tr>
<th>LRH value</th>
<th>TTS m = 10</th>
<th>TTS α = 0.01</th>
<th>TTS α = 0.005</th>
</tr>
</thead>
<tbody>
<tr>
<td>2778</td>
<td>2</td>
<td>2835</td>
<td>13</td>
</tr>
<tr>
<td>2578</td>
<td>2</td>
<td>2039</td>
<td>14</td>
</tr>
<tr>
<td>2666</td>
<td>2</td>
<td>2701</td>
<td>16</td>
</tr>
<tr>
<td>2713</td>
<td>2</td>
<td>2742</td>
<td>13</td>
</tr>
<tr>
<td>2855</td>
<td>2</td>
<td>2900</td>
<td>13</td>
</tr>
<tr>
<td>2992</td>
<td>3</td>
<td>2988</td>
<td>13</td>
</tr>
<tr>
<td>2749</td>
<td>2</td>
<td>2790</td>
<td>15</td>
</tr>
<tr>
<td>2874</td>
<td>2</td>
<td>2876</td>
<td>16</td>
</tr>
<tr>
<td>2745</td>
<td>2</td>
<td>2785</td>
<td>12</td>
</tr>
<tr>
<td>2837</td>
<td>2</td>
<td>2863</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 9: Performance of truncated tree search algorithms (D,20,50)

<table>
<thead>
<tr>
<th>LRH value</th>
<th>TTS m = 10</th>
<th>TTS α = 0.01</th>
<th>TTS α = 0.005</th>
</tr>
</thead>
<tbody>
<tr>
<td>2808</td>
<td>5</td>
<td>2844</td>
<td>9</td>
</tr>
<tr>
<td>2759</td>
<td>4</td>
<td>2815</td>
<td>8</td>
</tr>
<tr>
<td>2760</td>
<td>5</td>
<td>2814</td>
<td>10</td>
</tr>
<tr>
<td>2671</td>
<td>4</td>
<td>2733</td>
<td>9</td>
</tr>
<tr>
<td>2708</td>
<td>5</td>
<td>2873</td>
<td>10</td>
</tr>
<tr>
<td>2804</td>
<td>4</td>
<td>2850</td>
<td>10</td>
</tr>
<tr>
<td>2772</td>
<td>5</td>
<td>2813</td>
<td>9</td>
</tr>
<tr>
<td>2807</td>
<td>4</td>
<td>2763</td>
<td>9</td>
</tr>
<tr>
<td>2748</td>
<td>5</td>
<td>2853</td>
<td>9</td>
</tr>
<tr>
<td>2653</td>
<td>5</td>
<td>2705</td>
<td>10</td>
</tr>
</tbody>
</table>
Table 10: Performance of truncated tree search algorithms \((D,10,100)\)

<table>
<thead>
<tr>
<th>LRH</th>
<th>TTS(m = 10)</th>
<th>TTS(\alpha = 0.001)</th>
<th>TTS(\alpha = 0.005)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>cpu</td>
<td>value</td>
<td>cpu</td>
</tr>
<tr>
<td>5832</td>
<td>6</td>
<td>5864</td>
<td>249</td>
</tr>
<tr>
<td>5704</td>
<td>6</td>
<td>5737</td>
<td>237</td>
</tr>
<tr>
<td>5694</td>
<td>7</td>
<td>5752</td>
<td>192</td>
</tr>
<tr>
<td>5555</td>
<td>6</td>
<td>5575</td>
<td>192</td>
</tr>
<tr>
<td>5790</td>
<td>6</td>
<td>5860</td>
<td>236</td>
</tr>
<tr>
<td>5792</td>
<td>6</td>
<td>5774</td>
<td>224</td>
</tr>
<tr>
<td>5712</td>
<td>6</td>
<td>5745</td>
<td>244</td>
</tr>
<tr>
<td>5611</td>
<td>7</td>
<td>5631</td>
<td>209</td>
</tr>
<tr>
<td>5741</td>
<td>7</td>
<td>5790</td>
<td>219</td>
</tr>
<tr>
<td>5457</td>
<td>8</td>
<td>5470</td>
<td>207</td>
</tr>
</tbody>
</table>

Table 11: Performance of truncated tree search algorithms \((D,20,100)\)

<table>
<thead>
<tr>
<th>LRH</th>
<th>TTS(m = 10)</th>
<th>TTS(\alpha = 0.001)</th>
<th>TTS(\alpha = 0.005)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>cpu</td>
<td>value</td>
<td>cpu</td>
</tr>
<tr>
<td>5611</td>
<td>12</td>
<td>5710</td>
<td>99</td>
</tr>
<tr>
<td>5825</td>
<td>12</td>
<td>5867</td>
<td>93</td>
</tr>
<tr>
<td>5709</td>
<td>10</td>
<td>5801</td>
<td>93</td>
</tr>
<tr>
<td>5716</td>
<td>12</td>
<td>5753</td>
<td>92</td>
</tr>
<tr>
<td>5822</td>
<td>12</td>
<td>5840</td>
<td>91</td>
</tr>
<tr>
<td>5839</td>
<td>11</td>
<td>5891</td>
<td>99</td>
</tr>
<tr>
<td>5815</td>
<td>11</td>
<td>5907</td>
<td>98</td>
</tr>
<tr>
<td>5098</td>
<td>11</td>
<td>5755</td>
<td>92</td>
</tr>
<tr>
<td>5677</td>
<td>12</td>
<td>5738</td>
<td>91</td>
</tr>
<tr>
<td>5687</td>
<td>12</td>
<td>5745</td>
<td>97</td>
</tr>
</tbody>
</table>

23