J Heuristics (2010) 16: 239–258
DOI 10.1007/s10732-008-9100-4

Alternating control tree search for knapsack/covering problems

Lars Magnus Hvattum · Halvard Arntzen · Arne Løkketangen · Fred Glover

Received: 15 August 2007 / Revised: 7 November 2008 / Accepted: 7 November 2008 /
Published online: 26 November 2008

Abstract The *Multidimensional Knapsack/Covering Problem* (KCP) is a 0–1 *Integer Programming Problem* containing both knapsack and weighted covering constraints, subsuming the well-known *Multidimensional Knapsack Problem* (MKP) and the *Generalized (weighted) Covering Problem*. We propose an Alternating Control Tree Search (ACT) method for these problems that iteratively transfers control between the following three components: (1) ACT-1, a process that solves an LP relaxation of the current form of the KCP. (2) ACT-2, a method that partitions the variables according to 0, 1, and fractional values to create sub-problems that can be solved with relatively high efficiency. (3) ACT-3, an updating procedure that adjoins inequalities to produce successively more constrained versions of KCP, and in conjunction with the solution processes of ACT-1 and ACT-2, ensures finite convergence to optimality. The ACT method can also be used as a heuristic approach using early termination rules. Computational results show that the ACT-framework successfully enhances the performance of three widely different heuristics for the KCP. Our ACT-method involving scatter search performs better than any other known method on a large set of KCP-instances from the literature. The ACT-based methods are also found to be highly effective on the MKP.

L.M. Hvattum
Department of Industrial Economics and Technology Management, Norwegian University of Science and Technology, Trondheim, Norway
e-mail: Lars.M.Hvattum@himolde.no

H. Arntzen · A. Løkketangen (✉)
Molde University College, Molde, Norway
e-mail: Arne.Lokketangen@himolde.no

H. Arntzen
e-mail: Halvard.Arntzen@himolde.no

F. Glover
Leeds School of Business, University of Colorado, Boulder, CO, USA
e-mail: fred.glover@colorado.edu

Keywords Knapsack · Covering · Tree search · Heuristic

1 Introduction

In this work we present the Alternating Control Tree Search (ACT) method, and give an implementation for solving the *Multidimensional Knapsack/Covering Problem* (KCP). The KCP subsumes the classical 0–1 *Multidimensional Knapsack Problem* (MKP) and the *Generalized Covering Problem* (the latter being transformed into the former by complementing the variables), with associated applications to a broad range of problems, including capital budgeting (Lorie and Savage 1955; Manne and Markowitz 1957), cargo loading (Shih 1979), allocation (Gavish and Pirkul 1982), and cutting stock problems (Gilmore and Gomory 1966). When combining the knapsack and cover constraints, the resulting KCP problem has further applications in portfolio selection and capital budgeting (Beaujon et al. 2001), as well as in obnoxious and semi-obnoxious facility location (Cappanera 1999; Plastria 2001; Romero-Morales et al. 1997). Note that the KCP is sometimes referred to as the *Multidemand Multidimensional Knapsack Problem* (MDMKP).

We formulate the *Multidimensional Knapsack/Covering Problem* (KCP) as follows.

$$\max z = \sum_{j \in \mathbf{N}} c_j x_j$$

$$\text{s.t.} \quad \sum_{j \in \mathbf{N}} a_{ij} x_j \le b_i \quad \text{for } i \in \mathbf{M_A} \tag{1}$$

$$\sum_{j \in \mathbf{N}} a_{ij} x_j \ge b_i \quad \text{for } i \in \mathbf{M_B} \tag{2}$$

$$x_j \in \{0, 1\} \quad \text{for } j \in \mathbf{N} \tag{3}$$

Here, $\mathbf{N} = \{1, \dots, n\}$ is the set of variable indices, and $\mathbf{M_A} = \{1, 2, \dots, m\}$ and $\mathbf{M_B} = \{m+1, m+2, \dots, m+q\}$ are the sets of indices for the knapsack constraints and the cover constraints respectively. We assume that all coefficients, c_j, a_{ij}, and b_i are non-negative for all $j \in \mathbf{N}$ and $i \in \mathbf{M_A} \cup \mathbf{M_B}$.

The rest of the paper is structured as follows. In Sect. 2, we present the Alternating Control Tree Search (ACT) method for the *Multidimensional Knapsack/Covering Problem*. In Sect. 3 we discuss different alternatives for solving the subproblem associated with each iteration of ACT, leading to both heuristic and exact implementations. Computational results follow in Sect. 4. Finally, our conclusions are summarized in Sect. 5.

2 The alternating control tree search method

We propose an Alternating Control Tree Search (ACT) framework for the KCP, and show how to use various alternative solution methods for the KCP within the ACT-framework. The framework bears resemblance to the method used by Soyster et al.

(1978), later refined by Hanafi and Wilbaut (2006), but with some important differences. We note that the ACT-framework is quite general in the sense that it can be used both as an exact method as well as a heuristic algorithm, with the latter as an integrated combination of exact and heuristic methods (Puchinger and Raidl 2005).

The ACT-framework operates by transferring control successively to three components, ACT-1, ACT-2 and ACT-3, by the following design. We allow the possibility of knowing an initial integer feasible solution x^{LB} with associated value z^{LB} for the original KCP. If no starting integer feasible solution is known, z^{LB} is taken to be an arbitrary negative number. Supporting comments will subsequently be provided to elaborate on the method's summary description.

ACT-framework

ACT-1

1. Solve the LP relaxation LP(KCP) of the KCP to obtain a solution x' with value z'.
2. If LP(KCP) has no feasible solution, or if $z' \le z^{\mathrm{LB}}$, then the method terminates with one of the following conclusions.
   (a) If $z^{\mathrm{LB}} \ge 0$ then x^{LB} is optimal for the original KCP.
   (b) Otherwise, KCP is proven to have no feasible solution.

ACT-2

1. With reference to the solution x' of LP(KCP), let $\mathbf{N}^0(x') = \{j \mid x'_j = 0\}$, $\mathbf{N}^1(x') = \{j \mid x'_j = 1\}$, and $\mathbf{N}^{\mathrm{F}}(x') = \{j \mid 0 < x'_j < 1\}$.
2. Choose an assignment set $\mathbf{N}^{\#}(x')$ as a subset of $\mathbf{N}^0(x')$ or $\mathbf{N}^1(x')$. Define the sub-problem KCP-SUB to be the current KCP, subject to $x_j = x'_j$ for $j \in \mathbf{N}^{\#}(x')$ and

$$\sum_{j\in\mathbf{N}} c_j x_j \ge z^{\mathrm{LB}} + 1 \tag{4}$$

   (adjoining Eq. 4 to the inequalities 2).
3. Apply some method to solve KCP-SUB. If a feasible solution x^{SUB} is found, the solution yields $z^{\mathrm{SUB}} > z^{\mathrm{LB}}$, and this solution is recorded as the new x^{LB}, z^{LB}.

ACT-3

1. If the assignment set $\mathbf{N}^{\#}(x')$ is a subset of $\mathbf{N}^0(x')$, adjoin the following constraint to the system 2:

$$\sum_{j\in\mathbf{N}^{\#}(x')} x_j \ge 1 \tag{5}$$

   Or, alternatively, if $\mathbf{N}^{\#}(x')$ is a subset of $\mathbf{N}^1(x')$, adjoin the following constraint to the system 1:

$$\sum_{j\in\mathbf{N}^{\#}(x')} x_j \le |\mathbf{N}^{\#}(x')| - 1 \tag{6}$$

2. Return to ACT-1 to solve the resulting new version of KCP.

We clarify the operation of the method with the following preliminary observations.

Remark 1 The problem KCP-SUB has the form

$$\max z = \sum_{j \in \mathbf{N^{SUB}}} c_j x_j + c^{\mathrm{SUB}} \tag{7}$$

$$\text{s.t.} \quad \sum_{j \in \mathbf{N^{SUB}}} a_{ij} x_j \le b_i^{\mathrm{SUB}} \quad \text{for } i \in \mathbf{M_A} \tag{8}$$

$$\sum_{j \in \mathbf{N^{SUB}}} a_{ij} x_j \ge b_i^{\mathrm{SUB}} \quad \text{for } i \in \mathbf{M_B} \tag{9}$$

$$x_j \in \{0, 1\} \quad \text{for } j \in \mathbf{N^{SUB}} \tag{10}$$

where $c^{\mathrm{SUB}} = \sum_{j \in \mathbf{N}^{\#}(x')} c_j x'_j$, $b_i^{\mathrm{SUB}} = b_i - \sum_{j \in \mathbf{N}^{\#}(x')} a_{ij} x'_j$ for $i \in \mathbf{M_A} \cup \mathbf{M_B}$, and $\mathbf{N^{SUB}} = \mathbf{N} \setminus \mathbf{N}^{\#}(x')$.

Remark 2 The choice of the assignment set $\mathbf{N}^{\#}(x')$ as a subset of $\mathbf{N}^0(x')$ or $\mathbf{N}^1(x')$ in ACT-2, rather than as a subset of their union, assures that the inequalities adjoined in ACT-3 will fit in with the KCP structure. Our intention is to exploit the special structure that is maintained in both the master problem and KCP-SUB. In Soyster et al. (1978) and Hanafi and Wilbaut (2006), the assignment set is $\mathbf{N}^{\#}(x') = \mathbf{N}^0(x') \cup \mathbf{N}^1(x')$, which leads to smaller sub-problems, but a more general solver is required for handling the subsequent sub-problems.

Remark 3 The ACT method is clearly finite, since the inequalities adjoined in ACT-3 prevent the LP solution in ACT-1 from creating a situation where a currently selected assignment set $\mathbf{N}^{\#}(x')$ can duplicate any assignment set previously chosen. Likewise, the inclusion of these ACT-3 inequalities within KCP-SUB (as inherited from the current KCP) prevents the solution to this sub-problem from duplicating any solution previously obtained.

Remark 4 The ACT method leads to an exact algorithm for the KCP as long as an exact method is applied to solve KCP-SUB in ACT-2. Alternatively, the ACT method can take the role of a metaheuristic approach, by introducing early termination rules in ACT-1 and also in the solver of KCP-SUB. Also the method allows for using other heuristic solution methods to explore KCP-SUB. In these cases the method can no longer prove optimality or infeasibility upon termination.

Remark 5 It may happen that in some iteration, the solution x' to LP(KCP) is purely fractional. Then, the assignment set is empty and KCP-SUB equals KCP. Thus the subsolver in ACT-2 will work on the complete KCP rather than on a proper sub-problem. If the subsolver is an exact method, it will terminate with proven optimum or infeasibility of the KCP.

Remark 6 A reasonable initial choice for the assignment set $\mathbf{N}^{\#}(x')$ in many instances is to select $\mathbf{N}^{\#}(x') = \mathbf{N}^{1}(x')$, with the provision for selecting $\mathbf{N}^{\#}(x') = \mathbf{N}^{0}(x')$ if $\mathbf{N}^{1}(x')$ is somewhat smaller than $\mathbf{N}^{0}(x')$. On the other hand, choosing the assignment set $\mathbf{N}^{\#}(x')$ to be smaller than $\mathbf{N}^{1}(x')$ (or $\mathbf{N}^{0}(x')$) in ACT-2 permits the KCP-SUB problem to be more encompassing, and produces a stronger inequality to be added in ACT-3. (If $\mathbf{N}^{\#}(x')$ is chosen to be empty, then of course the problem KCP-SUB is simply the full KCP.) Opposing this, taking $\mathbf{N}^{\#}(x')$ to be larger has advantages not only for the ease of solving KCP-SUB, but more particularly for allowing the solution of LP(KCP) to guide the determination of new KCP-SUB problems.

Remark 7 By adding constraints 5 and 6, the ACT method essentially keeps an explicit store of previously visited partial assignments. Naturally, this causes worst-case exponential memory requirements.

Remark 8 Inequalities adjoined in ACT-3 may be selectively discarded from KCP as the method progresses. This may be done simply by removing older inequalities, or by reference to the dual weights received by these inequalities in the solution of LP(KCP). In particular, if an inequality receives a 0 dual weight (or receives such a weight on k successive iterations, for a small k value), then the inequality is a natural candidate to be discarded. Note however that removal of inequalities invalidates the finiteness property of the method, as the exhaustion of the solution space as described in Remark 3 is no longer guaranteed.

Remark 9 Under conditions where $\mathbf{N}^{0}(x')$ and $\mathbf{N}^{1}(x')$ are both small (or, in the extreme, empty), the method can embed a subset of $\mathbf{N}^{\mathrm{F}}(x')$) in $\mathbf{N}^{0}(x')$ or $\mathbf{N}^{1}(x')$, so that $\mathbf{N}^{\#}(x')$ can include some variables with fractional values from LP(KCP). This weakens the inequality introduced in ACT-3, but provides a heuristic supplement. Under such conditions, ACT-1 is only re-visited upon choosing $\mathbf{N}^{\#}(x')$ so that the inequality of ACT-3 is not satisfied by the most recent LP solution.

Remark 10 The subproblem, KCP-SUB, can frequently be reduced by preprocessing techniques (see e.g. Savelsbergh 1994), such as coefficient update routines and variable fixing based on reduced costs from the solution of the LP-relaxation.

One way of depicting the basic idea of the ACT method is shown in Fig. 1. The whole solution space for the KCP is here denoted by S. In ACT iteration i, let the subspace $S_i \subset S$ be the feasible region for the current KCP-SUB. As long as the ACT-3 inequalities are accumulated into the KCP and included in the KCP-SUB, the sets S_i will be disjoint. Moreover, a finite (but typically large) number of S_i will cover the entire feasible region for the original KCP (see Remark 3). Viewed in this way, the ACT method basically works by focusing the computational effort to smaller subspaces which are iteratively selected with the guidance from LP relaxations.

The ACT method exhibits some similarity to other frameworks designed to allow focused attention on selected subspaces. The Local Branching (LB) framework described in Fischetti and Lodi (2003) and the Relaxation Induced Neighborhood Search (RINS) from Danna et al. (2005) are noticeable examples. Both of these are

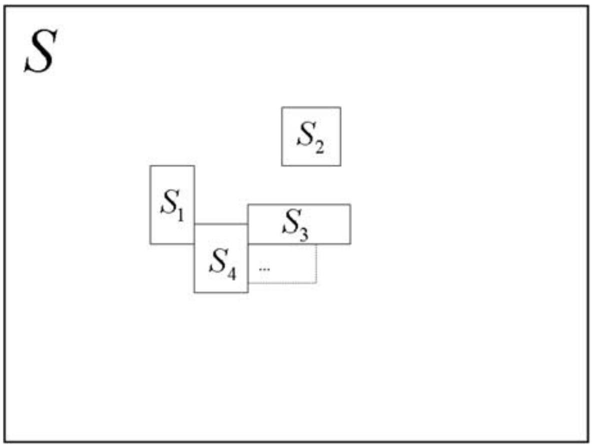

Fig. 1 Solution subspaces S_i related to the ACT-2 step

formulated for general MIP's. In the LB framework, the subspaces are defined as neighborhoods of the incumbent solution, using a generalized Hamming distance to define the neighborhood. Each subproblem is solved by a MIP solver, and when the incumbent solution is updated, a new subproblem emerges. The framework can yield either an exact method or a heuristic, depending on the approach taken when handling the subproblems.

The RINS framework is based on a branch-and-cut approach to solving a MIP. In certain nodes of the tree, one halts the branch-and-cut process temporarily. A subproblem is defined by fixing the variables that take on the same value in the incumbent solution and in the solution to the LP relaxation at the current node. This subproblem is explored, before resuming the branch-and-cut process. The underlying assumption is that solving the subproblem is likely to provide a better incumbent solution faster than the ordinary branch-and-cut routine would otherwise do. If the incumbent solution is improved, this will be helpful when the branch-and-cut is resumed. In addition, when the MIP must be explored within a fixed time limit, it can be beneficial to increase the focus on early improvement in the branch-and-bound, which is just what RINS is supposed to do.

These two approaches are different from the ACT method in several ways. ACT is specially tailored for the KCP problem structure, while the others are more general. The subspace selection in LB and RINS is based on having a feasible solution of the MIP, thus the frameworks can not be properly put to work before the first feasible solution is found. In contrast, ACT starts immediately from the solution of the LP relaxation to the original KCP.

A different type of cuts from those generated by ACT, are the conflict cuts introduced in Achterberg (2007). These cuts are designed to cut off infeasible regions earlier (Achterberg reports a general 18 percent reduction in the number of nodes for general MIP's in his tests). These aspects would be important in particular when ACT is used as a complete solver.

The idea of focusing the search on small subspaces at the time goes back at least to Glover (1977), with the concept of consistent and strongly determined variables, and is also pointed out many places in Glover and Laguna (1997). This idea is also used successfully in Holmberg and Yuan (2000), Sellmann et al. (2002) and Gomes and Sellmann (2004).

Finally a word on the type of relaxation used for search guidance. We are using the LP relaxation for partioning the variables, as this is readily available. Other relaxations might be more efficient, as stated by Gomes et al. (2006). They show that in their case (MAX SAT) the predictive power of variable fixing based on semi-definite relaxations is much higher than that of LP relaxations. This is also an interesting avenue for further investigations.

Additional options available for applying the ACT method, and those we have elected to implement for solving the KCP, are described in Sects. 3 and 4. We now turn to describe different solution methods suitable for solving KCP-SUB in ACT-2.

3 Solving the subproblem

As noted in Sect. 2, the subproblem, KCP-SUB, has the same structure as the KCP itself. As a consequence, any method for solving the KCP can be used as a subsolver in ACT-2. In this section we outline several methods for solving the KCP, based on using different subsolvers for KCP-SUB.

3.1 CPLEX

A natural benchmark for measuring the performance of our ACT-implementations is to solve the instances using a standard, commercial IP-solver. The solver we use for this purpose is CPLEX (version 9.0), with standard settings. We will refer to the method as **CPLEX**. In addition, it is clear that **CPLEX** is a candidate for solving the KCP-SUB problem. Hence, we obtain a method **ACT-CPLEX-EXACT**, which implements the ACT-framework, using **CPLEX** as a subsolver.

Based on some initial testing to calibrate the method, we found the following settings to work well for **ACT-CPLEX-EXACT**. Firstly, we always select the larger of $\mathbf{N}^0(x')$ and $\mathbf{N}^1(x')$ to be the assignment set $\mathbf{N}^\#(x')$ in ACT-2. The initial testing further indicated that keeping the canonical cuts added in ACT-3 also in the KCP-SUB leads to improved solution speeds, and any attempt at removing these over time, based on the ideas presented in Remark 8, decreases the solution quality. We do not add any preprocessing of the KCP-SUB (as suggested in Remark 10), since **CPLEX** already includes many preprocessing techniques. Lastly, the **CPLEX**-method, as used for the KCP-SUB, was instructed to focus on feasibility first, which seems to improve the overall results slightly. Note that these specifications apply to **CPLEX** as used in **ACT-CPLEX-EXACT**, and not to the stand-alone **CPLEX**-method.

Even though both **CPLEX** and **ACT-CPLEX-EXACT** are, in principle, able to prove optimality, in practice the available time is usually too short to get such a proof. In these cases, one might consider looking for heuristic solutions only, and hence we consider another way of including CPLEX within the ACT-framework, creating an implementation we refer to as **ACT-CPLEX**. In **ACT-CPLEX** we use the solver CPLEX as in **ACT-CPLEX-EXACT** with the exception that it is terminated after a given time limit has expired. In addition, to make **ACT-CPLEX** more easily comparable to the methods described in Sects. 3.2 and 3.3, the following adjustments are made. First, an additional preprocessing step for each subproblem to potentially reduce the size of KCP-SUB before applying the subsolver is included. The preprocessing consists of removing redundant constraints, locking variables (based on logical tests, as well as on reduced costs from the associated Linear Programming relaxation), and updating the constraint coefficients (see Savelsbergh 1994 for some standard ideas). Second, the canonical cuts added in ACT-3, of type (Eq. 5) or (Eq. 6), are not included in the subproblem when tackled by the subsolver.

3.2 Tabu search

The option of looking for heuristic solutions also opens up the possibility of considering metaheuristic solution methods. We now move on to present a Tabu Search method for the KCP, which can be used either as a stand-alone method, or as a subsolver within the ACT-framework.

A Tabu Search (Adaptive Memory Search) for the KCP problem is described in detail in Arntzen et al. (2006), and only the main structure is outlined here. The search was initially developed for problem instances with both knapsack and cover constraints, and due to the difficulty of finding feasible solutions for such instances (and since the search space can be separated into different feasible regions, Cappanera and Trubian 2005), it is important for a Local Search based method to be able to search in both the feasible space and the infeasible space. Thus, the Tabu Search method, here referred to as **TS**, is based on an oscillation around the feasibility boundaries by coordinating the interplay between changes in the objective function values and changes in primal feasibility. The search has the following components.

1. The *starting solution* is based on a random assignment of values to the variables, with equal probabilities of assigning 0 or 1. This solution is usually infeasible.
2. A *move* is the flip of a variable. A flip means assigning the opposite value to a variable.
3. The *search neighborhood* is the set of solutions reachable in one flip.
4. *Move evaluation* is based on both the change in objective function value, and the change in amount of infeasibilities.
5. The *move selection* is greedy.
6. Simple *tabu* and *aspiration criteria* are enabled.
7. The *stopping criterion* is a simple time limit.

For further details, and a summary of parameter settings, we refer the reader to Arntzen et al. (2006), whereas a general introduction to Tabu Search can be found in Glover and Laguna (1997). We now describe how **TS** can be applied as a subsolver in the ACT-framework, either alone as in **ACT-TS** or together with **CPLEX** as in **ACT-CPLEX-TS**.

For **ACT-TS**, where **TS** is used as the only subsolver, the settings of **TS** remain the same as when **TS** is used as a stand-alone method. The same changes made in the ACT-framework for **ACT-CPLEX** compared to the **ACT-CPLEX-EXACT**-method is applied. This refers to adding a preprocessing step to reduce the size of KCP-SUB before applying the Tabu Search, and to exclude the canonical cuts added in ACT-3, of type (Eq. 5) or (Eq. 6), from the subproblem when tackled by the subsolver. Again, the subsolver is set to terminate after a given amount of time.

In **ACT-CPLEX-TS**, both **CPLEX** and **TS** are applied, in sequence, as solvers for the KCP-SUB. **CPLEX** is used first, and if this method returns a proven optimum, **TS** is not used. If, after a set time limit, **CPLEX** has not proved optimality, **TS** is started. The best solution to KCP-SUB, found by either method, is returned to the ACT-framework.

3.3 Scatter search

The final solution method for the KCP presented here is based on Scatter Search (see, e.g. Laguna and Martí 2003 for an introduction), and is presented with details in Hvattum and Løkketangen (2007). As for the Tabu Search in Sect. 3.2, this method can be applied both alone and as a subsolver in the ACT-framework.

Five different Scatter Search implementations are given in Hvattum and Løkketangen (2007), and the one used in this work was entitled **SB SS** (we refer to it here simply as **SS**). This variant seemed to be the most robust of the choices examined in Hvattum and Løkketangen (2007), being the only one to find feasible solutions to all of the problem instances for which it was tested. We now summarize briefly the components of this Scatter Search implementation.

1. The *Diversification Generation* method creates an initial pool of randomly generated solutions.
2. The *Improvement* method is based on what we call a jagged Local Search, which is a Local Search that continues as long as the best neighbor is better than the previous solution (unlike a standard Local Search, which continues as long as the best neighbor is better than the current solution). The neighborhood structure includes both flip-moves as well as double-flip moves (where two variables are flipped at once).
3. The *Reference Set Update* method creates and maintains a reference set with a mix of good solutions and diverse solutions as well as solutions that have contributed to the creation of other good solutions.
4. The *Subset Generation* method generates all subsets of size two of the reference set, with the added condition that at least one of the solutions in the subset was added to the reference set in the previous iteration.
5. Finally, the *Solution Combination* method is based around a generalization of a score based method, as proposed in Laguna and Martí (2003).

Again, we leave further details and information about parameter settings to Hvattum and Løkketangen (2007). For usage within the ACT-framework, we use the same settings as for **ACT-TS** and **ACT-CPLEX-TS**, but with **SS** instead of **TS**, to create **ACT-SS** and **ACT-CPLEX-SS**.

4 Computational results and analyzes

All computational testing is done on PCs with Pentium IV 2.4 GHz processors and 1 GB of RAM. In the following we report and discuss the results for the following nine methods.

- **CPLEX** is the use of the standard settings for CPLEX version 9.0.
- **TS** is the Tabu Search as described in Sect. 3.2.
- **SS** is the Scatter Search as described in Sect. 3.3.
- **ACT-CPLEX-EXACT** is an implementation of the ACT-framework as an exact solution method, using CPLEX to solve each of the subproblems.
- **ACT-CPLEX** is an implementation of the ACT-framework as an heuristic solution method, using CPLEX with a given time limit to solve each of the subproblems.
- **ACT-TS** is an implementation of the ACT-framework as a heuristic solution method, using **TS** to produce solutions to each of the subproblems.
- **ACT-SS** is similar to **ACT-TS**, except that **SS** is used to produce solutions to the subproblems.
- **ACT-CPLEX-TS** is also similar to **ACT-TS**, but uses a combination of **CPLEX** and **TS** to produce solutions to the subproblems.
- **ACT-CPLEX-SS** is similar to **ACT-CPLEX-TS**, except that **CPLEX** and **SS** is used to produce solutions to the subproblems.

Preliminary testing gave us the following timing scheme. All the methods are limited to a running time of at most 1 hour, except when used as subsolvers within another method. Based on some calibration runs, the following schemes were used to distribute the computational effort among the components of the different methods. In **ACT-CPLEX-EXACT**, the subsolver (CPLEX) was allowed to run until optimality was proved or until the overall time limit was reached. For both **ACT-TS** and **ACT-SS**, the **TS** and **SS** was run for 100 seconds on each subproblem. The same time limit was used for **CPLEX** when applied within **ACT-CPLEX**. When using both **CPLEX** and **TS** or **SS** as subsolvers, **CPLEX** was first allowed to execute for 25 seconds (or until optimality was proved, whichever occurred first). If optimality for the subproblem was not proved, **TS** or **SS** was run for 100 seconds respectively in **ACT-CPLEX-TS** and **ACT-CPLEX-SS**.

The test instances used in the computational results reported here reflect the fact that the KCP subsumes the MKP. From Chu and Beasley (1998) we have 9 classes, with 30 instances each, for the MKP. These have $n = 100$, $n = 250$, or $n = 500$ variables, and $m = 5$, $m = 10$, or $m = 30$ knapsack constraints. Another class (with results reported in Vasquez and Hao 2001) contains 11 MKP-instances with between $n = 100$ and $n = 2500$ variables, and from $m = 15$ to $m = 100$ knapsack constraints, which gives a total of 281 MKP instances. In addition, results are reported for a set of 405 instances with both knapsack and cover constraints, from Cappanera and Trubian (2005). This set is a subset of the instances presented in Cappanera and Trubian (2005), obtained by disregarding instances with negative coefficients in the objective function. The KCP instances have the same number of variables and knapsack constraints as the MKP instances, but have in addition $q = 1$, $q = \lfloor \frac{m}{2} \rfloor$, or $q = m$ cover constraints (15 instances of each type).

Before embarking on the discussion of the results, we emphasize that we start with relative comparisons. These may graphically give the impression that the differences in performance between some methods are larger than they actually are. The absolute results are presented in Table 2, towards the end.

Our discussion around the results start with Fig. 2, which shows, for each method, the ratio of problems for which the method has the best found solution as a function

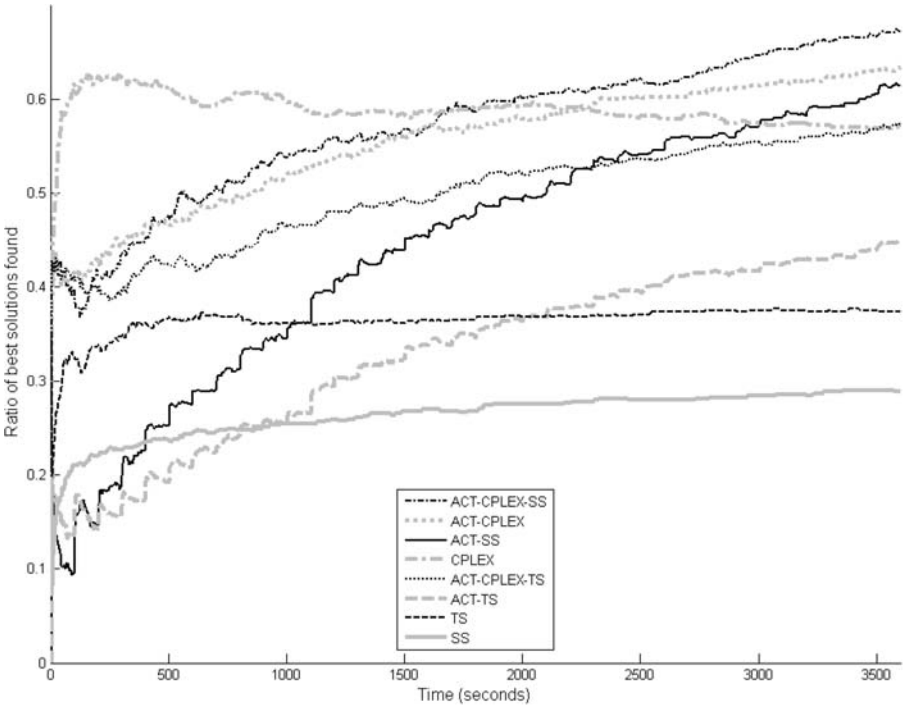

Fig. 2 Ratio of best solutions as a function of time for the different solution methods, on all instances

of time. For example, after 300 seconds **CPLEX** has found at least as good results as any other method for about 62% of the instances, and after 3600 seconds this ratio has decreased to just under 58% for **CPLEX**.

In this and the following figures, all methods are applied to the same set of instances. In Fig. 2 the set of instances is the complete collection of MKP and KCP instances, while subsequent figures show results for different subsets this collection.

Actually, **CPLEX** is an exception by having a deteriorating performance; i.e., its relative performance becomes worse over time. In terms of the number of best solutions found, **CPLEX** loses its lead at around 1700 seconds, and the best method from then on is the **ACT-CPLEX-SS** method (having about 67% of the best results after 3600 seconds). After one hour, **ACT-CPLEX** and **ACT-SS** is also better than **CPLEX**. for **ACT-CPLEX-TS** the performance is equal to that of **CPLEX** whereas the other methods are inferior.

A general finding, which is illustrated in Fig. 2, is that using the heuristic solution methods (**SS** and **TS**) within the ACT-framework gives better results than using them alone. We can also note that the same is not true for **CPLEX** when used as an exact subsolver (**ACT-CPLEX-EXACT** is not shown in Fig. 2, but final results are listed in Table 1), since **ACT-CPLEX-EXACT** gives a smaller number of best solutions (about 48% versus 57%) than **CPLEX**. Also of interest is the fact that ACT-CPLEX-EXACT finds more feasible solutions than CPLEX. However, there is also a difference in the usage, as **CPLEX** as a subsolver is not terminated until optimality

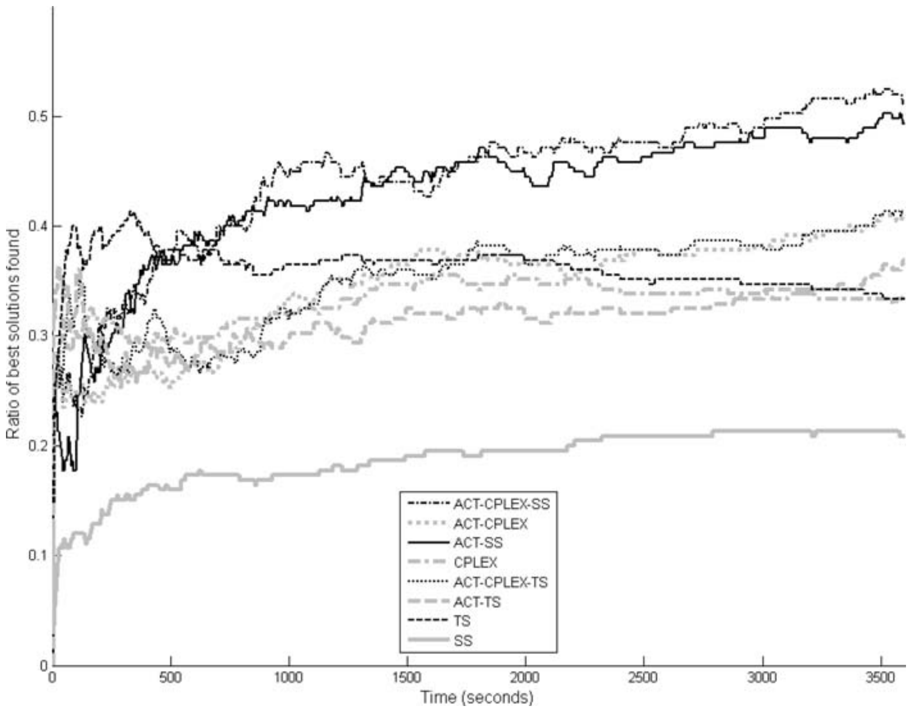

Fig. 3 Ratio of best solutions as a function of time for the different solution methods for the problem instances with 30 knapsack constraints

of the subproblem has been proven in **ACT-CPLEX-EXACT**, while **SS** and **TS** both are terminated after a given amount of time. A consequence is that for some of the difficult problems, **ACT-CPLEX-EXACT** actually spends all the time allotted having **CPLEX** working on the first subproblem. This may prevent this method from finding a good (or any) feasible solution, which would only appear in the investigation of a later subproblem. To complete the picture, when **CPLEX** is applied as a heuristic subsolver, as in **ACT-CPLEX**, it does indeed perform better than as a stand-alone method.

The same type of graph is plotted in Fig. 3, but only for the subset of instances with 30 knapsack constraints (and 0, 1, 15, or 30 cover constraints). The best performance is now by **ACT-CPLEX-SS** and **ACT-SS**, and in this subset also the **ACT-CPLEX-TS** as well as **ACT-CPLEX** performs better than **CPLEX**. The results obtained with **ACT-TS**, **ACT-CPLEX-EXACT**, and **TS** are comparable to **CPLEX** while **SS** is a bit behind. It thus seems that the best problem instances to be handled by **CPLEX** are those with few constraints.

Figure 4 shows only the instances with 500 variables. Again, **CPLEX** seems best early in the runs, but is then overtaken by **ACT-CPLEX-SS**, **ACT-CPLEX** and later also by **ACT-SS**. A notable feature with respect to the instances with 500 variables is that the lone **SS** method does not find any of the best solutions (at any point in time). This is consistent with the findings in Hvattum and Løkketangen (2007) that Scatter

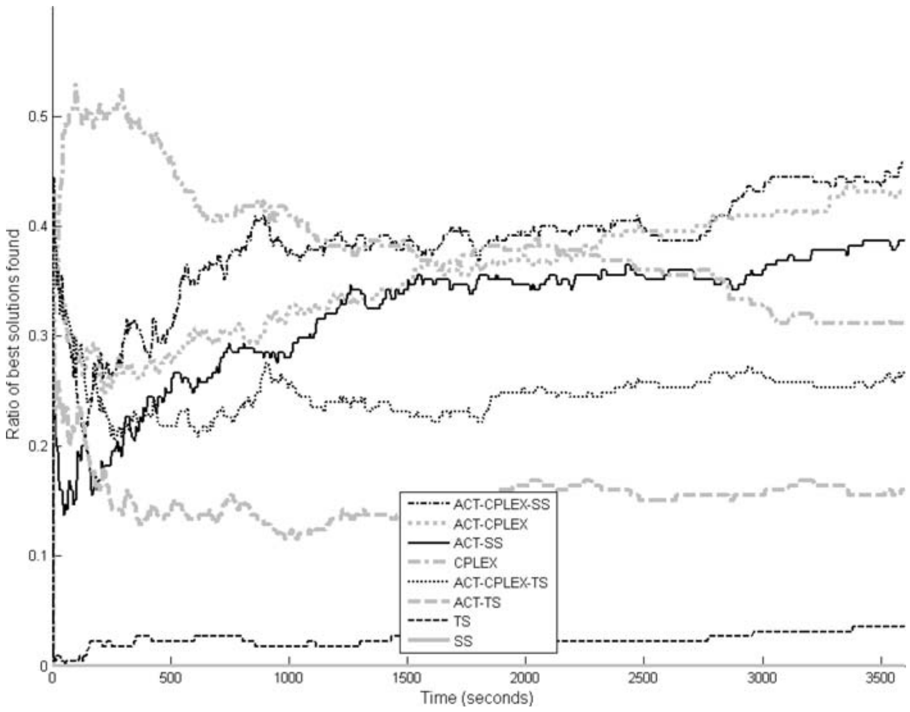

Fig. 4 Ratio of best solutions as a function of time for the different solution methods for the problem instances with 500 variables

Search (in the versions employed there) needs a lot of time to find good solutions, especially on the larger instances. It seems that the partitioning of the search space as done within the ACT-framework is very beneficial for Scatter Search, enabling **ACT-SS** to perform well even for these larger problem instances. Since the subproblems are significantly smaller than the original problem, Scatter Search can find reasonably good solutions to the subproblems quite quickly, even though the full problem would take a long time to solve similarly well.

Finally, Fig. 5 shows results for the subset of instances with at least one cover constraint, whereas Fig. 6 shows results for the instances without any cover constraints. The most notable feature is that **CPLEX** performs very well on the pure MKP problems, especially early in the runs. There could be several explanations for this. First of all, the other methods are to some extent tuned to perform well on the general KCP, with both **TS** and **SS** having search strategies and parameters that reflect the existence of cover constraints. In addition, the overall ACT-framework is constructed around the interplay between knapsack and cover constraints. Finally, we expect that the ease of finding feasible solutions for the MKP problems helps the branch-and-bound approach of **CPLEX** to cut off parts of the search space more efficiently than for the problem instances with cover constraints. It is also interesting to observe in Figs. 5 and 6 that more or less all of the ACT-based methods are gaining steadily on **CPLEX** and the other non-ACT methods. The expected outcome of longer running

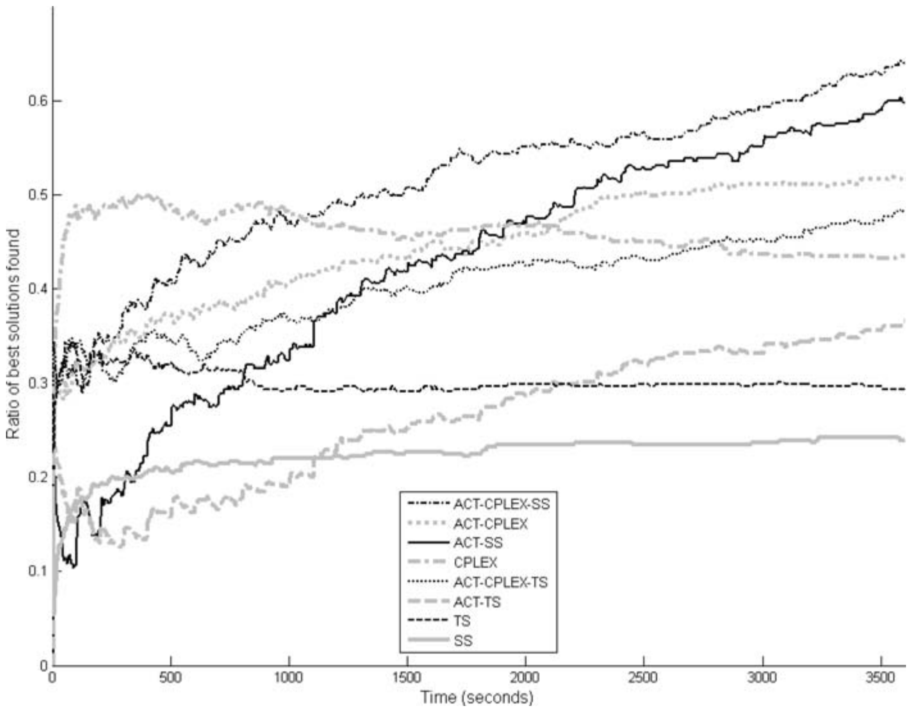

Fig. 5 Ratio of best solutions as a function of time for the different solution methods for the instances with both cover and knapsack constraints

times would thus be to see even more favorable relative results for the ACT-based methods.

An additional remark is appropriate. Figures 2–6 seem to indicate that there is a huge difference between the good and the bad methods. E.g., in Fig. 5, **SS** finds the best solution for only about 24% of the instances, whereas **ACT-CPLEX-SS** finds the best solution for more than 64% of the instances. However, if we compare the value of the best solution found by **SS** with the best value found overall, the difference is typically much less than 1%, and for the MKP instances of Chu and Beasley (1998) the difference is on average 0.057%. This remark also applies to Table 1, as discussed next.

The results in Table 1 are interpreted as follows. For each problem class and each method a number is given that represents the relative performance of the method for that particular class. The number is scaled so that it lies in [0, 1], being 1 for the method that has the best average objective function value for the class, and being 0 for the method that has the worst average objective function value. This means that, for example, for class KCP 100-5, where **TS** has an average objective function value within 0.034% of the best method, it is still assigned a value of 0, on account of being the method with the worst average objective function value. The classes are listed based on type (KCP has both knapsack and cover constraints, MKP has only knapsack constraints) and size (**n–m** gives the number of variables and the number of

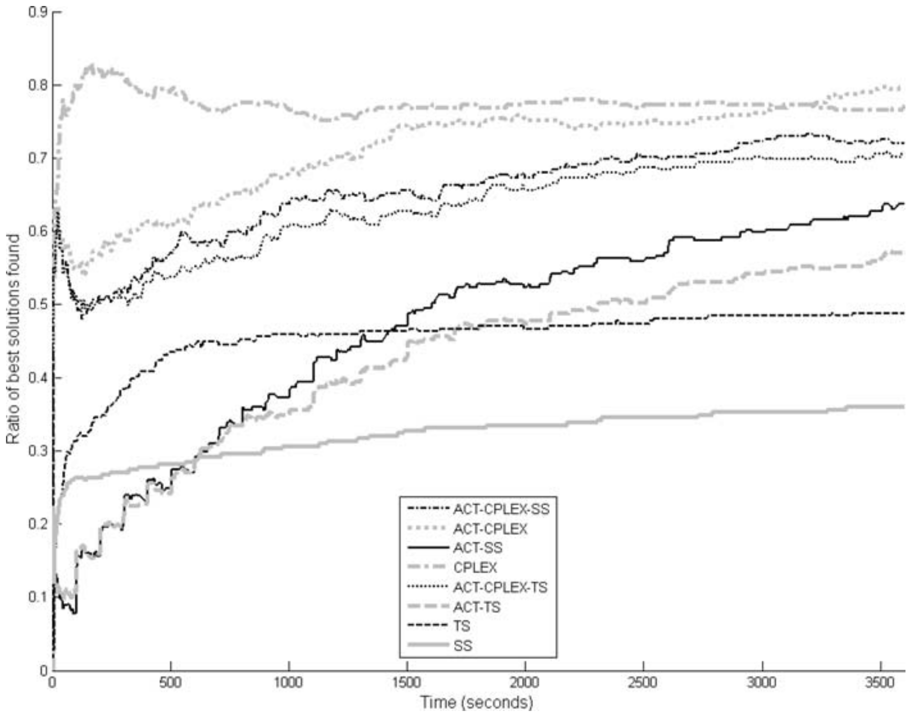

Fig. 6 Ratio of best solutions as a function of time for the different solution methods for the MKP instances

knapsack constraints). The column # gives the number of instances in each class. If the method fails to find feasible solutions to some of the instances in a class, instead of reporting the scaled average objective function of the solved instances, we report the ratio of solutions found in brackets.

Summarizing the results of Table 1, the main impression is similar to that of Figs. 2–6. Overall, **ACT-CPLEX-SS** and **ACT-CPLEX** give the best results, but **ACT-SS**, **CPLEX**, **ACT-CPLEX-EXACT**, and **ACT-CPLEX-TS** all perform nearly as well. However, it is necessary to stress that the average results (in the row entitled AVERAGE*) exclude the three KCP-classes with $m = 30$ knapsack constraints (since these are the classes where some methods fail to produce feasible solutions to all the instances). Excluding these latter three classes from the average gives a favorable bias to **CPLEX**, **ACT-CPLEX** and **ACT-CPLEX-EXACT** in particular. It should be noted that all methods except **TS** and **SS** fail to find feasible solutions for some of the KCP-instances. As Table 1 shows though, **CPLEX** and **ACT-CPLEX-EXACT** fail to find feasible solutions in more instances than the other methods (with 33 and 25 failed instances, respectively).

Table 2 gives average values of the best solutions found for each method and each class. All numbers are rounded to the nearest integer. If a method does not find feasible solutions to all instances in a class, we report as in Table 1. The best average results for each class are emphasized by boldface numbers. The table confirms the

Table 1 Summary of relative performance within each problem instance class

	n–m	#	ACT-CPLEX-SS	ACT-CPLEX	ACT-SS	CPLEX	ACT-CPLEX-TS	ACT-CPLEX-EXACT	ACT-TS	TS	SS
KCP	100–5	45	1.000	1.000	1.000	1.000	1.000	1.000	0.723	0.000	0.830
KCP	100–10	45	0.991	1.000	0.630	0.931	0.944	0.991	0.472	0.551	0.000
KCP	100–30	45	(39/45)	(34/45)	(38/45)	(27/45)	(39/45)	(33/45)	(38/45)	1.000	0.000
KCP	250–5	45	0.725	1.000	0.688	0.750	0.688	0.738	0.550	0.238	0.000
KCP	250–10	45	0.972	0.905	1.000	0.804	0.927	0.642	0.877	0.799	0.000
KCP	250–30	45	1.000	(43/45)	0.996	(35/45)	0.768	(37/45)	0.761	0.537	0.000
KCP	500–5	45	1.000	0.989	0.984	0.973	0.908	0.876	0.811	0.249	0.000
KCP	500–10	45	1.000	0.932	0.991	0.911	0.911	0.789	0.864	0.591	0.000
KCP	500–30	45	1.000	(40/45)	0.961	(40/45)	0.916	(40/45)	0.880	0.510	0.000
MKP	100–5	30	1.000	1.000	1.000	1.000	1.000	1.000	0.783	0.000	1.000
MKP	100–10	30	1.000	1.000	0.452	1.000	1.000	1.000	0.452	1.000	0.000
MKP	100–30	30	1.000	1.000	1.000	1.000	1.000	1.000	0.977	1.000	0.000
MKP	250–5	30	0.929	0.929	0.786	1.000	0.929	1.000	0.500	0.000	0.071
MKP	250–10	30	0.982	1.000	0.930	1.000	0.982	0.912	0.895	0.825	0.000
MKP	250–30	30	0.953	1.000	0.915	1.000	0.969	0.713	0.961	0.860	0.000
MKP	500–5	30	1.000	1.000	0.945	1.000	1.000	0.964	0.873	0.691	0.000
MKP	500–10	30	0.968	0.989	0.952	1.000	0.952	0.866	0.957	0.710	0.000
MKP	500–30	30	0.953	1.000	0.908	0.945	0.955	0.834	0.874	0.716	0.000
GK-MKP		11	0.998	1.000	0.972	0.996	0.999	0.998	0.945	0.000	0.015
UNSOLVED		686	6	18	7	33	6	25	7	0	0
AVERAGE*		551	0.963	0.981	0.881	0.945	0.938	0.882	0.766	0.514	0.126

Table 2 Summary of absolute performance within each problem instance class

	n–m	#	ACT-CPLEX-SS	ACT-CPLEX	ACT-SS	CPLEX	ACT-CPLEX-TS	ACT-CPLEX-EXACT	ACT-TS	TS	SS
KCP	100–5	45	**46520**	**46520**	**46520**	**46520**	**46520**	**46520**	46516	46504	46518
KCP	100–10	45	**38754**	**38754**	38746	38753	38753	**38754**	38743	38745	38733
KCP	100–30	45	(39/45)	(34/45)	(38/45)	(27/45)	(39/45)	(33/45)	(38/45)	**29960**	29889
KCP	250–5	45	127881	**127903**	127878	127883	127878	127882	127867	127842	127823
KCP	250–10	45	101734	101722	**101739**	101704	101726	101675	101717	101703	101560
KCP	250–30	45	**85461**	(43/45)	**85461**	(35/45)	85402	(37/45)	85400	85342	85205
KCP	500–5	45	**264499**	264497	264496	264494	264482	264476	264464	264360	264314
KCP	500–10	45	**222759**	222729	222755	222720	222720	222666	222699	222579	222319
KCP	500–30	45	**179446**	(40/45)	179416	(40/45)	179381	(40/45)	179353	179067	178673
MKP	100–5	30	**42640**	**42640**	**42640**	**42640**	**42640**	**42640**	**42640**	42638	**42640**
MKP	100–10	30	**41606**	**41606**	41604	**41606**	**41606**	**41606**	41604	**41606**	41602
MKP	100–30	30	**40768**	**40768**	**40768**	**40768**	**40768**	**40768**	40767	**40768**	40763
MKP	250–5	30	107088	107088	107086	**107089**	107088	**107089**	107082	107075	107076
MKP	250–10	30	106364	**106365**	106361	**106365**	106364	106360	106359	106355	106308
MKP	250–30	30	104698	**104704**	104693	**104704**	104700	104667	104699	104686	104575
MKP	500–5	30	**214165**	**214165**	214162	**214165**	**214165**	214163	214158	214148	214110
MKP	500–10	30	212824	212828	212821	**212830**	212821	212805	212822	212776	212644
MKP	500–30	30	211379	**211399**	211360	211376	211380	211329	211346	211279	210977
GK-MKP		11	25726	**25727**	25720	25726	25726	25726	25715	25506	25510
UNSOLVED		686	6	18	7	33	6	25	7	**0**	**0**
AVERAGE*		551	**124910**	**124910**	124907	124905	124905	124890	124896	124861	124788

general pattern of results from Table 1. In most cases the differences between the best method and the runner-ups within the classes are marginal.

As a curiosity, we mention that two new best solutions were found on the 500-variable MKP instances. The best known results on these 90 instances are reported in Vasquez and Vimont (2005). For problem instance number 13 in the class with $n = 500$ and $m = 10$ of MKP-instances, where the previous best result was 217806, **CPLEX** finds a solution with a value of 217847. For problem instance number 24 in the same class, the improvement is from 300757 to 300784, which is found by **ACT-CPLEX-SS** and **ACT-SS**. In addition, for the MKP problem with 500 variables the previous best known solutions are replicated for 35 out of the 90 problem instances (though not by all of the methods). One should note here that the running time in Vasquez and Vimont (2005) is 50–100 hours per instance on average, whereas our results were achieved within one hour of running time (per method). Considering **ACT-CPLEX-SS** alone, this method run for one hour replicates best known solutions for 16 of the 90 instances and improves the result on one instance. Looking at the results for **ACT-CPLEX**, which appears to be the most effective of our methods for the MKP, we find that this method alone replicates 23 best known results. Comparing the results for **ACT-CPLEX** directly to those reported in Hanafi and Wilbaut (2006) we get better solutions on 32 instances and worse on 27 instances. Our results on these 90 instances are thus on par with those from Hanafi and Wilbaut (2006), which were obtained with somewhat shorter running times.

5 Conclusions

We have proposed, implemented, and tested the Alternating Control Tree Search (ACT) method, which is a framework that can be used both for exact and for heuristic solution methods. Four different solution methods for the *Multidimensional Knapsack/Covering Problem* (KCP) was implemented both as stand-alone methods and as subsolvers within the ACT-framework.

Among the stand-alone methods, which do not integrate ACT with other procedures, **CPLEX** found more best solutions than the other methods (implemented in isolation), though, in reverse, it also failed to find feasible solutions to more of the problems than any of the other methods. For the methods based on ACT, **ACT-CPLEX-SS** and **ACT-CPLEX** were the best and outperformed **CPLEX**, except for **ACT-CPLEX-SS** on instances of the pure *Multidimensional Knapsack Problem* (MKP). We note that in general, using a heuristic method within the ACT-framework (as in **ACT-SS** and **ACT-TS**) gives better results than using the heuristic alone (as in **SS** and **TS**). The same observation can be made when **CPLEX** is used as a heuristic, for **ACT-CPLEX**. Our testing furthermore shows that **TS** tends to give somewhat better results than **SS**, when used alone. When used with the ACT, the **SS** is set to work on smaller subproblems and it seems to work better than the **TS** in this role.

The ACT is similar to the method examined in Soyster et al. (1978) and Hanafi and Wilbaut (2006), but differs in trying to exploit the special structure of the KCP while Soyster et al. (1978) and Hanafi and Wilbaut (2006) focus on a general 0–1 Integer Programming formulation, while performing computation tests only for the

MKP. Moreover, Soyster et al. (1978) and Hanafi and Wilbaut (2006) claims that their method is suitable for problems with few constraints, whereas our findings indicate that the ACT-framework is indeed successful for problems with many constraints (see Fig. 3 compared to Fig. 2).

We do reach the same conclusion as in Hanafi and Wilbaut (2006), however, in that the framework is best suited as a heuristic mechanism. This is mainly due to the fact that the progression in lowering the upper bounds produced in ACT-1 becomes increasingly slow as more iterations are performed.

Acknowledgements We thank two anonymous referees for comments and suggestions that improved on a preliminary version of this paper.

References

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

Arntzen, H., Hvattum, L.M., Løkketangen, A.: Adaptive memory search for multidemand multidimensional knapsack problems. Comput. Oper. Res. **33**, 2508–2525 (2006)

Beaujon, G.J., Marin, S.P., McDonald, G.C.: Balancing and optimizing a portfolio of r&d projects. Nav. Res. Logist. **48**, 18–40 (2001)

Cappanera, P.: Discrete facility location and routing of obnoxious facilities. PhD thesis, University of Milano (1999)

Cappanera, P., Trubian, M.: A local-search-based heuristic for the demand-constrained multidimensional knapsack problem. INFORMS J. Comput. **17**, 82–98 (2005)

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

Danna, E., Rothberg, E., Pape, C.L.: Exploring relaxation induced neighborhoods to improve MIP solutions. Math. Program. **102**(1), 71–90 (2005)

Fischetti, M., Lodi, A.: Local branching. Math. Program. **98**(1), 23–47 (2003)

Gavish, B., Pirkul, H.: Allocation of databases and processors in a distributed computing system. In: Akoka, J. (ed.) Management of Distributed Data Processing, pp. 215–231. North-Holland, Amsterdam (1982)

Gilmore, P.C., Gomory, R.E.: The theory and computation of knapsack functions. Oper. Res. **14**, 1045–1075 (1966)

Glover, F.: Heuristics for integer programming using surrogate constraints. Decis. Sci. **8**(1), 156–166 (1977)

Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Boston (1997)

Gomes, C., Sellmann, M.: Streamlined constraint reasoning. In: Proceedings, CPAIOR 2004 (2004)

Gomes, C., van Hoeve, W., Leahu, L.: The power of semidefinite programming relaxations for MAX-SAT. In: Proceedings, CPAIOR 2006, pp. 104–118 (2006)

Hanafi, S., Wilbaut, C.: Improved convergent heuristic for 0–1 mixed integer programming. Research Report, University of Valenciennes (2006)

Holmberg, K., Yuan, D.: A Lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. Oper. Res. **48**(3), 461–481 (2000)

Hvattum, L.M., Løkketangen, A.: Experiments using scatter search for the multidemand multidimensional knapsack problem. In: Doerner, K.F., et al. (eds.) Metaheuristics: Progress in Complex Systems Optimization. Operations Research/Computer Science Interfaces, vol. 39, pp. 3–24. Springer, Berlin (2007)

Laguna, M., Martí, R.: Scatter Search: Methodology and Implementations in C. Kluwer Academic, Dordrecht (2003)

Lorie, J., Savage, L.: Three problems in capital rationing. J. Bus. **28**, 229–239 (1955)

Manne, A., Markowitz, H.: On the solution of discrete programming problems. Econometrica **25**, 85–110 (1957)

Plastria, F.: Static competitive facility location: an overview of optimization approaches. Eur. J. Oper. Res. **129**, 461–470 (2001)

Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: Mira, J., Álvarez, J.R. (eds.) Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II. Lecture Notes in Computer Science, vol. 3562, pp. 41–53. Springer, Berlin (2005)

Romero-Morales, D., Carrizosa, E., Conde, E.: Semi-obnoxious location models: a global optimization approach. Eur. J. Oper. Res. **102**, 295–301 (1997)

Savelsbergh, M.W.P.: Preprocessing and probing for mixed integer programming problems. ORSA J. Comput. **6**, 445–454 (1994)

Sellmann, M., Kliewer, G., Koberstein, A.: Lagrangian cardinality cuts and variable fixing for capacitated network. In: Proceedings of the Tenth Annual European Symposium on Algorithms, pp. 845–858 (2002)

Shih, W.: A branch and bound method for the multiconstraint zero-one knapsack problem. J. Oper. Res. Soc. **30**, 369–378 (1979)

Soyster, A.L., Lev, B., Slivka, W.: Zero-one programming with many variables and few constraints. Eur. J. Oper. Res. **2**, 195–201 (1978)

Vasquez, M., Hao, J.-K.: A hybrid approach for the 0–1 multidimensional knapsack problem. In: Proceedings of the International Joint Conference on Artificial Intelligence 2001, pp. 328–333 (2001)

Vasquez, M., Vimont, Y.: Improved results on the 0–1 multidimensional knapsack problem. Eur. J. Oper. Res. **165**, 70–81 (2005)