1 Exact, Metaheuristic and Hybrid Approaches to Multi-dimensional Knapsack Problems

J. E. GALLARDO, C. COTTA, A. J. FERNÁNDEZ,

Dpto. de Lenguajes y Ciencias de la Computación, E.T.S.I. Informática
Campus Teatinos 29071, Málaga, SPAIN
e-mail: {pepeg,ccottap,afdez}@lcc.uma.es

1.1 INTRODUCTION

This chapter reviews our recent work on applying hybrid collaborative techniques that integrate Branch and Bound (B&B) and Memetic Algorithms (MAs) in order design effective heuristics for the Multidimensional Knapsack Problem (MKP). To this end, let us recall that Branch and Bound (B&B) [1] is an exact algorithm for finding optimal solutions to combinatorial problems, that, basically works by producing convergent lower and upper bounds for the optimal solution using an implicit enumeration scheme. A different approach to optimization is provided by evolutionary algorithms [2, 3, 4] (EAs). These are powerful heuristics for optimization problems based on principles of natural evolution, namely adaptation and survival of the fittest. Starting from a *population* of randomly generated *individuals* (representing solutions), a process consisting of *selection*, (promising solutions are chosen from the population) *reproduction* (new solutions are created by combining selected ones) and *replacement* (some solutions are replaced by the new ones) is repeated. A *fitness* function measuring the quality of the solution is used to guide the process.

A key aspect of EAs is robustness, meaning that they can be deployed on a wide range of problems. However, it has been shown that some kind of domain knowledge has to be incorporated into EAs for them to be competitive with other domain specific optimization techniques [5, 6, 7]. A very successful approach to achieve this knowledge-augmentation are Memetic Algorithms (MAs) [8, 9, 10, 11], that integrate domain-specific heuristics into the EA. In this chapter hybridizations of an MAs with B&B are presented. These hybridizations are aimed to combining the search capabilities of both algorithms in a synergetic way.

The remainder of the chapter is organized as follows: in the rest of this section, general descriptions of Branch and Bound and Memetic Algorithms will be given. In Section 1.2, the *Multidimensional Knapsack Problem* will be described along with Branch and Bound and a Memetic Algorithm to tackle it. Then, Section 1.3 presents our hybrid proposals integrating both approaches. Subsequently, Section 1.4 shows and analyzes the empirical results obtained by the application of each of the described approaches on different instances of the benchmark. Finally, Section 1.5 provides the conclusions and outlines ideas for future work.

1.1.1 Branch and Bound

The Branch and Bound algorithm (B&B) is a very general exact technique, first proposed in [12], that can be used to solve combinatorial optimization problems. B&B is basically an enumeration approach that prunes the non-promising regions of the search space [1]. For this purpose, the algorithm uses two key ingredients: a *branching rule*, that provides a way of dividing a region of the search space into smaller regions (ideally partitions of the original one), and a way to calculate an *upper bound* (assuming a maximization problem) on the best solution that can be attained in a region of the search space.

A very general pseudo code for this algorithm is shown in Figure 1.1. The algorithm maintains the best solution found so far (*sol* in the pseudo code), usually named *the incumbent* one. B&B starts by considering the whole search space of the problem (S). This is split into several subspaces (denoted by set C in the algorithm) using the branching rule. If a subspace is trivially solved and its value improves on the incumbent solution, the latter is updated. Otherwise, a bound for each subspace is calculated. If the bound of one subspace is worse than the incumbent solution, this part of the search space can safely be eliminated (this is called *pruning*). Otherwise, the process goes on until all subspaces are either solved or pruned, and the final value of the incumbent solution is returned as an optimal solution to the problem.

The efficiency of the algorithm relies on the effectiveness of the branching rule and the accuracy of the bounds. If these ingredients of the algorithm are not well defined, the technique degenerates into an exhaustive inefficient search. On one side, an effective branching rule will avoid revisiting the same regions of the search space. One the other side, tight bounds will allow more pruning. Usually, it is difficult to find tight bounds that are not too expensive computationally, and a balance has to be found between the amount of pruning achieved and the computational complexity of the bounds. In any

Branch and Bound Algorithm

```
1:  $open := S$
2:  $sol := null$ /* whose value is assumed $-\infty$ */
3:  **while** $open \neq \emptyset$ **do**
4:    **select** s **from** $open$
5:    $open := open \setminus \{s\}$
6:    $C :=$ **branch on** $s$
7:    **for** $c \in C$ **do**
8:      **if** is Solved($c$) **then**
9:        **if** value($c$) > value($sol$) **then** $sol := c$ **end if**
10:     **else if** Upper Bound($c$) > value($sol$) **then**
11:       $open := open \cup \{c\}$
12:     **end if**
13:   **end for**
14: **end while**
15: **return** $sol$
```

Fig. 1.1 Generic template for Branch and Bound algorithm.

case, these techniques cannot practically be used on large instances for many combinatorial optimization problems (COPs).

Note that the search performed by the algorithm can be represented as a tree traversed in a certain way (that depends on the order in which subspaces are introduced and extracted in set *open*). If a depth-first strategy is used, the memory required grows linearly with the depth of the tree; hence large problems can be considered. However, the time-consumption can be excessive. On the other hand, a best-first strategy minimizes the number of nodes explored, but the size of the search tree (that is, the number of nodes kept for latter expansion) will grow exponentially in general. A third option is to use a breadth-first traversal (i.e., every node in a level is explored before moving on to the next). In principle, this option would have the drawbacks of the previous two strategies, unless a heuristic choice is made: to keep at each level only the best nodes (according to some *quality* measure). This implies sacrificing exactness, but provides a very effective heuristic search approach. The name *Beam Search* (BS) has been coined to denote this strategy [13, 14]. As such, BS algorithms are incomplete derivatives of B&B algorithms, and are thus heuristic methods. A generic description of BS is depicted in Figure 1.2. Essentially, BS works by extending every partial solution from a set $\mathcal{B}$ (called the *beam*) in all possible ways. Each new partial solution so generated is stored in a set $\mathcal{B}$'. When all solutions in $\mathcal{B}$ have been processed, the algorithm constructs a new beam by selecting the best up to k_{bw} (called the

Beam Search Algorithm

```
1:  $sol := null$ /* whose value is assumed $-\infty$ */
2:  $\mathcal{B} := \{ () \}$
3:  **while** $\mathcal{B} \neq \emptyset$ **do**
4:     $\mathcal{B}' := \emptyset$
5:     **for** $s \in \mathcal{B}$ **do**
6:        **for** $c \in$ Children of($s$) **do**
7:           **if** is Completed($c$) **then**
8:              **if** value($c$) $>$ value($sol$) **then** $sol := c$ **end if**
9:           **else if** Upper Bound($c$) $> sol$ **then**
10:             $\mathcal{B}' := \mathcal{B}' \cup \{ c \}$
11:          **end if**
12:       **end for**
13:    **end for**
14:    $\mathcal{B} :=$ **select** best $k_{bw}$ nodes **from** $\mathcal{B}$'
15: **end while**
16: **return** $sol$
    **end function**
```

Fig. 1.2 Generic template for Beam Search algorithm.

beam width) solutions from $\mathcal{B}$'. Clearly, a way of estimating the quality of partial solutions, such as an upper bound or an heuristic, is needed for this.

One context in which B&B is frequently used is Integer Programming (IP) [15]. IP is a generalization of Linear Programming (LP) [16], so let us first describe this latter problem. In LP there are *decision variables*, *constraints* and an *objective function.* Variables take real values, and both the objective function and the constraints must be expressed as a series of linear expressions in the decision variables. The goal is to find an assignment for the variables maximizing (or minimizing) the objective function. Linear programs can be used to model many practical problems, and nowadays solvers can find optimal solutions to problems with hundred of thousands of constraints and variables in reasonable time. The first proposed algorithm to solve LPs was the *Simplex Method* devised by Dantzig in 1947 [17], that performs very well in practice although its performance is exponential in the worst case. Afterwards, other methods that run in polynomial time have been proposed like *Interior Points* methods [18]. In practice, the Simplex Algorithm works very well in most cases, and other approaches are only useful for very large instances.

An Integer Program is a Linear Program in which some or all variables are restricted to take integer values. This *small* change in the formulation increases considerably the number of problems that can be modeled, but also increases dramatically the difficulty of solving problems. One approach for solving IPs is to use a B&B algorithm that uses solutions to *Linear Relaxations* as bounds. These LP-relaxations are obtained by removing the integral requirements on decision variables and can efficiently be solved using any LP method. Assuming a maximization problem, it follows that a solution to its LP-relaxation is an upper bound for the solution of the original problem. In order to branch, the search space is usually partitioned into two parts by choosing an integer variable x_i that has fractional value v_i^f in the LP-relaxed solution, and introducing two branches with the additional constraints $x_i \leq \lfloor v_i^f \rfloor$ and $x_i \geq \lceil v_i^f \rceil$. One common heuristic is to choose as branching variable the one with fractional part closest to 0.5, although commercial IP solvers use more sophisticated techniques.

1.1.2 Memetic Algorithms

The need to exploit problem-knowledge in heuristics has been repeatedly shown in theory and in practice [7, 19, 5, 6]. Different attempts have been made to answer this need; *Memetic Algorithms* [8, 9, 10, 11] (MAs) is probably one of the most successful to date [20].

The adjective 'memetic' comes from the term 'meme', coined by R. Dawkins [21] to denote an analogous to the *gene* in the context of cultural evolution. As Evolutionary Algorithms, MAs are also population based metaheuristics. The main difference is that the components of the population (named *agents* in the MAs terminology) are not passive entities. These agents are active entities that cooperate and compete in order to find improved solutions.

There are many possible ways to implement MAs. The most common implementation consists of combining an EA with a procedure to perform local search that is usually done after evaluation, although it must be noted however that the MA paradigm does not simply reduce itself to this particular scheme. According to Eiben and Smith [4], Figure 1.3 shows places were problem specific knowledge can be incorporated. Some of the possibilities are:

- During the initialization of the population some heuristic method may be used to generate high quality initial solutions. One example of this kind of techniques will be elaborated in this chapter: namely using a variant of a B&B algorithm to periodically initialize the population of a MA with the aim of improving its performance.

- Recombination or mutation operators can be intelligently designed so that specific problem knowledge is used in order to improve the offspring.

- Problem knowledge can be incorporated in the genotype to phenotype mapping present in many EAs, like when repairing an unfeasible solution. This technique is used in the MA designed by Chu and Beasley [22] for the MKP, that is described in Section 1.2.3.

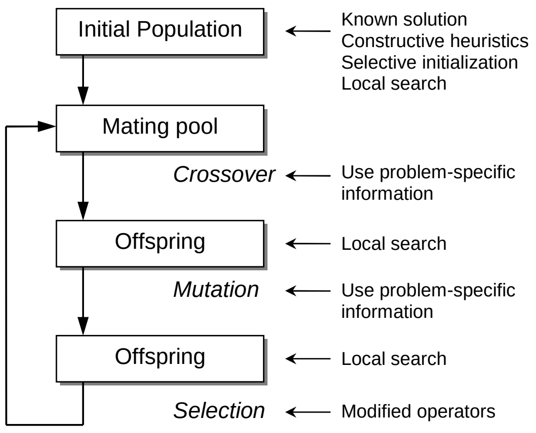

Fig. 1.3 Places to incorporate problem knowledge within an evolutionary algorithm, according to [4].

In addition to other domains, MAs have proven very successful across a wide range of combinatorial optimization problems, where they are state-of-the-art approaches for many problems. For a comprehensive bibliography, the reader may consult [23][1].

1.2 THE MULTIDIMENSIONAL KNAPSACK PROBLEM

In this section, we will review B&B Algorithms and a Memetic Algorithm that have been proposed to tackle the MKP. Let us first introduce the problem.

1.2.1 Description of the Problem

The *Multidimensional Knapsack Problem* (MKP) is a generalization of the classical *knapsack problem* (KP). In the KP, there is a knapsack with an upper weight limit b, and a collection of n items with different values p_j and weights r_j. The problem is to choose the collection of items which gives the highest total value without exceeding the weight limit of the knapsack.

In the MKP, m knapsacks with different weight limits b_i must be filled with the same items. Furthermore, these items have a different weight r_{ij} for each knapsack i. The objective is to find a set of objects with maximal profit such that the capacities of the knapsacks is not exceeded. Formally, the problem can be formulated as:

[1]See also the Memetic Algorithms home page `http://www.densis.fee.unicamp.br/~moscato/memetic_home.html`

$$\text{maximise} \qquad \sum_{j=1}^{n} p_j x_j, \tag{1.1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \tag{1.2}$$

$$x_j \in \{0, 1\}, \qquad j = 1, \ldots, n. \tag{1.3}$$

where x is an n-ary binary vector such that $x_j = 1$ if i-th object is included in the knapsacks or 0 otherwise.

Each of the m constraints in Eq. (1.2) is called a knapsack constraint. The problem can be seen as a general statement of any zero-one integer programming problem with non-negative coefficients. Many practical problems can be formulated as an instance of the MKP, for example, the capital budgeting problem, project selection and capital investment, budget control, and numerous loading problems (see e.g. [24]).

Although the classical Knapsack Problem is weakly NP-hard and admits a fully polynomial time approximation scheme (FPTAS) [25], this is not the case for the MKP which is much harder even for $m = 2$, and does not admit a FPTAS unless P=NP [26]. On the other hand, there exists a polynomial time approximation scheme (PTAS) with a running time of $O(n^{\lceil m/\varepsilon \rceil - m})$ [27, 28], that implies a non-polynomial increase in running time with respect to the accuracy sought.

1.2.2 Branch and Bound algorithms for the MKP

As stated earlier, a complete solution for an instance of the MKP with n objects will be represented by an n-ary binary vector v such that $v_i = 1$ if i-th object is included in the solution or 0 otherwise. The natural way of devising a B&B algorithm for the MKP is to start with a totally unspecified solution (a solution for which the inclusion or exclusion of any object is unknown), and to progressively fix the state of each object. This will lead to a branching rule that generates two new nodes from each partial node in the B&B tree by including or excluding a new object in the corresponding solution.

A way of estimating an upper bound for partial solutions is needed, so that non-promising partial solutions can be eliminated from the B&B queue. For this purpose, we will consider two possibilities. In the first one, when the j-th object is included in the solution, the lower bound for this partial solution is increased with the corresponding profit p_j (and the remaining available space is decreased by r_{ij} in each knapsack i), whereas the upper bound is decreased by p_j when the item is excluded.

The second possibility is to carry out an standard LP exploration of the search tree for this kind of problems, as it was described in Section 1.1.1 (see also [29]). More precisely, the linear relaxation of each node is solved (i.e., the problem corresponding to unfixed variables is solved, assuming variables can take fractional values in the interval [0,1]) using linear-programming (LP) techniques, like the Simplex method [17]. If all variables take integral values, the subproblem is solved. This is not generally the case though, and some variables are non-integer in the LP-relaxed solution; in the latter situation, the variable whose value is closest to 0.5 is selected, and two subproblems are generated, fixing this variable to 0 or to 1 respectively. The LP-relaxed value of the node is used as its upper-bound, so that nodes whose value is below the best-known solution can be pruned from the search tree.

1.2.3 Chu and Beasley's MA for the MKP

The MKP has been tackled via EAs in many works, e.g., [22, 30, 31, 32, 33, 34]. Among these, the EA developed by Chu and Beasley [22] remains as one of the cutting-edge approaches for solving the MKP. This EA uses the natural codification of solutions, namely binary n-dimensional strings $\vec{x}$, representing the incidence vector of a subset S of objects on the universal set O (i.e., $(x_j = 1) \Leftrightarrow o_j \in S$).

Of course, infeasible solutions might be encoded in this way, and this has to be considered in the EA. Typically, these situations can be solved in three ways: (i) allowing the generation of infeasible solutions and penalizing accordingly, (ii) using a repairing mechanism for mapping infeasible solutions to feasible solutions, and (iii) defining appropriate operators and/or problem representation to avoid the generation of infeasible solutions. Chu and Beasley's approach is based on the second solution, that is, a Lamarckian repairing mechanism is used (see [35] for a comparison of Lamarckian and Baldwinian repair mechanisms for the MKP). To do so, an initial pre-processing of the problem instance is performed off-line. The goal is to obtain a heuristic precedence order among variables: they are ordered by decreasing *pseudo-utility* values : $u_j = \frac{p_j}{\sum_{i=1}^{m} a_i r_{ij}}$, where we set the surrogate multipliers a_i to the dual variable values of the solution of the LP-relaxation of the problem (see [22] for details). Variables near the front of this ordered list are more likely to be included in feasible solutions (and analogously, variables near the end of the list are more likely to be excluded from feasible solutions). More precisely , whenever an infeasible solution is obtained, variables are set to zero in increasing order of pseudo-utility until feasibility is restored. After this, feasible solutions are improved by setting variables to one in decreasing order of pseudo-utility (as long as no constraint is violated). This way, the repairing algorithm can actually be regarded as a (deterministic) local improvement procedure, and hence this EA certainly qualifies as a MA. Since this MA just explores the feasible portion of the search space, the fitness function can be readily defined as $f(\vec{x}) = \sum_{j=1}^{n} p_j x_j$.

1.3 HYBRID MODELS

In this section, we present hybrid models that integrates an MA with B&B. Our aim is to combine the advantages of both approaches and, at the same time, avoid (or at least minimize) their drawbacks working alone. Firstly, in the following subsection, we briefly discuss some related works existing in the literature regarding the hybridization of exact techniques and metaheuristics.

1.3.1 Hybridizing exact and metaheuristic techniques

Although exact and heuristics methods are two very different ways of tacking COPs, they can be combined in hybrid approaches with the aim of combining synergistically their characteristics. This has been recognized by many researchers, and as a result, many proposals have emerged in the last years. Some authors have reviewed and classified the literature on this topic.

In [36], a classification of algorithms that combine local search and exact methods is presented. They focus on hybrid algorithms in which exact methods are used to strengthen local search.

In [37], an extensive compendium of exact/metaheuristics hybrid approaches for different COPs is presented. Among the different families of problems reviewed are mixed integer programming, graph coloring, frequency assignment, partitioning, maximum independent sets, maximum clique, traveling salesman, vehicle routing, packing, job-shop scheduling problems, etc.

Puchinger and Raidl [38] classification of exact and metaheuristics hybrid techniques is possibly the more general one. There are two main categories in their classification: *Collaborative* and *Integrative* combinations. In the following, we describe in detail each one.

1.3.1.1 Collaborative combinations This class includes hybrid algorithms were the exact and metaheuristic methods exchange information, but none of them is a subordinate of the other. As both algorithms have to be executed, two cases can be considered:

- *Sequential execution*, in which one of the algorithms is completely executed before the other. For instance, one technique can act as a kind of preprocessing for the other or the result of one algorithm can be used to initialize the other.
- *Parallel or intertwined execution*, where both techniques are executed simultaneously, either in parallel (i.e., running at the same time on different processors) or in an intertwined way by alternating between both algorithms.

One example in the first group is the hybrid algorithm proposed by Vasquez and Hao [39] to tackle the MKP, that combines Linear Programming and Tabu Search. Their central hypothesis is that the neighborhood of a solution to a relaxed version of the problem contains high quality solutions to the original problem. They use a two phase algorithm that first solves exactly a relaxation of the problem, and afterwards explores carefully and efficiently its neighborhood. For the first phase, they use the Simplex method, whereas for the second one they use a Tabu Search procedure. This hybrid algorithm produces excellent results for large and very large instances of the problem.

An example of parallel execution is the one presented by Denzinger *et al* [40]. It is based on a multi-agent model for obtaining cooperation between search systems with different search paradigms. The basic schema consisted of teams with a number of agents subjected to the same search paradigm and that exchanged certain class of information. To demonstrate the viability of the approach, a GA and B&B based system for a Job-Shop Scheduling Problem was described. Here, the GA and B&B agents exchanged only information in form of solutions whereas the B&B agents could also exchange information in form of closed subtrees. As results of the cooperation, better solutions were found given a timeout.

This chapter presents proposals in the second group, namely hybridizations of a B&B variant (like using *depth first* strategy or a Beam Search algorithm) – and a Memetic Algorithm, that are executed in an intertwined way.

1.3.1.2 Integrative combinations Here, one of the technique uses the other one for some purpose, and so the first one acts as a master whereas the second one behaves as a subordinate component of the other.

Again, two cases can be considered. The first one consists of incorporating an exact algorithm into a metaheuristic. One example is the MA for the MKP by Chu and Beasley [22] (described in detail in Section 1.2.3).

Cotta *et al* [41] presented a framework for the hybridization along the lines initially sketched in [42], i.e., based on using the B&B algorithm as a recombination operator embedded in the EA. This hybrid operator is used for recombination: it intelligently explores the possible children of solutions being recombined, providing the best possible outcome. The resulting hybrid algorithm provided better results than pure EAs in several problems where a full B&B exploration was unpractical on its own.

Puchinger and Raidl [43] represented another attempt to incorporate exact methods in metaheuristics. This work considered different heuristics algorithms for a real world glass cutting problem and a combined GA and B&B approach was proposed. The GA used an order-based representation that was decoded with a greedy heuristic. Incorporating B&B in the decoding for occasionally (with a certain probability) locally optimizing subpatterns turned out to increase the solution quality in a few cases.

The other possibility for integrative combinations is to incorporate a metaheuristics into an exact algorithm. One example is the hybrid algorithm combining Genetic Algorithms and Integer Programming B&B approaches to solve MAX-SAT problems described in [44]. This hybrid algorithm gathered information during the run of a linear programmming-based B&B algorithm, and used it to build the population of an EA population. The EA was eventually activated, and the best solution found was used to inject new nodes in the B&B search tree. The hybrid algorithm was run until the search tree was exhausted, and hence it is an exact approach. However, in some cases it expands more nodes than the B&B algorithm alone. A different approach has been tackled recently in [45] where a metaheuristic is used for strategic guidance of an exact method of B&B. The idea was to use a genetic programming (GP) model to obtain improved node selection strategies within B&B for solving mixed integer problems. The information collected from the B&B after operating a certain amount of time is used as a training set for GP which is run to find a node selection strategy more adequate for the specific problem. Then a new application of the B&B used this improved strategy. The idea was very interesting although it is not mature and requires further research to obtain more general conclusions.

Cotta *et al* [42] used a problem-specific B&B approach for the Traveling Salesman Problem based on 1-trees and the Lagrangean relaxation [46], and made use of an EA to provide bounds in order to guide the B&B search. More specifically, two different approaches for the integration were analyzed. In the first model, the genetic algorithm played the role of master and the B&B was incorporated as a slave. The primary idea was to build a hybrid recombination operator based in the B&B philosophy. More precisely, the B&B was used in order to build the best possible tour within the (Hamiltonian) subgraph defined by the union of edges in the parents. This recombination procedure was costly, but provided better results than blind edge recombination. The second model proposed consisted of executing in parallel the B&B algorithm with a certain number of EAs which generated a number of different high-quality solutions. The diversity provided by the independent EAs contributed to make that edges suitable to be part of the optimal solution were likely included in some individuals, and non-suitable edges were unlikely taken into account. Despite these approaches showed encouraging results, the work in [42] described only preliminary results.

1.3.2 Our Hybrid Proposals

One way to do the integration of evolutionary techniques and B&B models is via a *direct collaboration* that consists of letting both techniques work alone in parallel (i.e., let both processes perform independently), that is, at the same level. Both processes will share the solution. There are two ways of obtaining a benefit of this parallel execution:

- The B&B can use the lower bound provided by the EA to purge the problem queue, deleting those problems whose upper bound is smaller than the one obtained by the EA.
- The B&B can inject information about more promising regions of the search space into the EA population in order to guide the EA search.

In our hybrid approach, a single solution is shared among the EA and B&B algorithms that are executed in an interleaved way. Whenever one of the algorithms finds a better approximation, it updates the solution and yields control to the other algorithm.

Two implementation of this scheme will be considered. In the first one [47], the hybrid algorithm starts by running the EA in order to obtain a first approximation to the solution. In this initial phase, the population is randomly initialized and the EA executed until the solution is not improved for a certain number of iterations. This approximation can be later used by the B&B algorithm to purge the problem queue. No information from the B&B algorithm is incorporated in this initial phase of the EA, in order to avoid the injection of high-valued building blocks that could affect diversity, polarizing further evolution.

Afterwards, the B&B algorithm is executed. Whenever a new solution is found, it is incorporated into the EA population (replacing the worst individual), the B&B phase is paused and the EA is run to stabilization. Periodically, pending nodes in the B&B queue are incorporated into the EA population. Since these are partial solutions and the EA population consists of full solutions, they are completed and corrected using the repair operator. The intention of this transfer is to direct the EA to these regions of the search space. Recall that the nodes in the queue represent the subset of the search space still unexplored. Hence, the EA is used for finding probably good solutions in this region. Upon finding an improved lower bound (or upon stabilization of the EA, in case no improvement is found), control is returned to the B&B, hopefully with an improved lower bound. This process is repeated until the search tree is exhausted, or a time limit is reached. The hybrid is then an anytime algorithm that provides both a quasi-optimal solution, and an indication of the maximum distance to the optimum.

One interesting peculiarity of BS is that it works by extending in parallel a set of different partial solutions in several possible ways. For this reason, BS is a particulary suitable tree search method to be used in a hybrid collaborative framework, as it can be used to provide periodically diverse promising partial solutions to a population based search method such as a MA. We have precisely used this approach in our second implementation [48, 49], and a general description of the resulting algorithm is given in Figure 1.4.

The algorithm starts by executing BS for l_0 levels of the search tree. Afterwards, the MA and BS are interleaved until a termination condition is reached. Every time the MA is run, its population is initialized using the nodes in the BS queue. Usually, the size of the BS queue will be larger than the MA population size, so some criteria must be used to select a subset from the queue. For instance, this criteria may be selecting the best nodes according to some measure of quality or selecting a subset from the BS queue that provides high diversity. Let us note, that nodes in the BS queue represent schemata, i.e, they are partial solutions in which some genes are fixed but others are indeterminate, so they must first be converted to full solutions in

General description of hybrid algorithm

```
 1: for $l_0$ levels do run BS
 2: do
 3:     select popsize nodes from problem queue
 4:     initialize MA population with selected nodes
 5:     run MA
 6:     if MA solution better than BS solution then
 7:         let BS solution := MA solution
 8:     for $l$ levels do run BS
 9: until timeout or tree-exhausted
10: return BS solution
```

Fig. 1.4 The general template for hybrid of Beam Search and MA.

a problem dependent way. This is another aspect that must be considered when instantiating the general schema for different combinatorial problems.

Upon stabilization of the MA, control is returned to the B&B algorithm. The lower bound for the optimal solution obtained by the MA is then compared to the current incumbent in the B&B, updating the latter if necessary. This may lead to new pruned branches in the BS tree. Subsequently, BS is executed for descending l levels of the search tree .This process is repeated until the search tree is exhausted or a time limit is reached.

1.4 EXPERIMENTAL RESULTS

We tested our algorithms with problems available at the OR-library [50] maintained by Beasley. We took two instances per problem set. Each problem set is characterized by a number m of constraints (or knapsacks), a number n of items and a *tightness ratio*, $0 \leq \alpha \leq 1$. The closer to 0 the tightness ratio the more constrained the instance. All algorithms were coded in C, and all tests were carried on a Pentium IV PC (1700MHz and 256MB of main memory).

1.4.1 Results for first model

In this section we describe the results obtained by the first hybrid model described in Section 1.3.2. The B&B algorithm explores the search tree in a depth first way, and uses the first simple upper bound described in Section 1.2.2. A single execution for each instance was performed for the B&B methods whereas ten runs were carried out for the EA and hybrid algorithms. The algorithms were run for 600 seconds in all cases. For the EA and the hybrid algorithm, population size was fixed at 100 individuals that were initialized with random feasible solutions. Mutation probability was set to 2

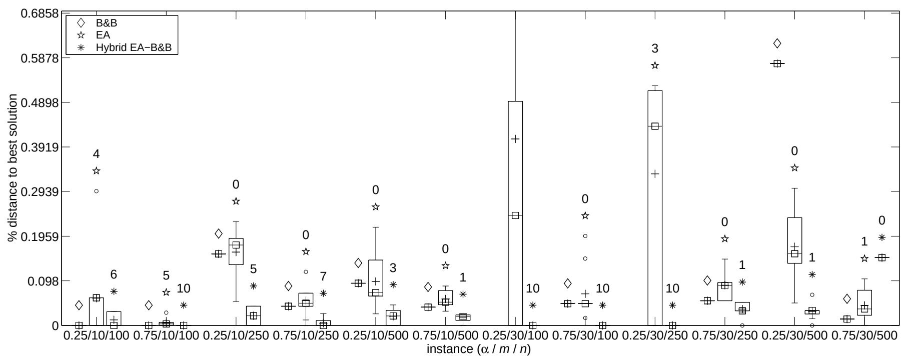

Fig. 1.5 Results of the B&B algorithm, the EA, and the first hybrid model for problem instances of different number of items (n), knapsacks (m), and tightness ratio (α). Boxplots show the relative distance to the best solution found by the three algorithms. In all box plots, the figure above indicates the number of runs (out of 10) leading to the best solution. A + sign indicates the mean of the distribution, whereas a □ marks its median. Boxes comprise the second and third quartiles of the distribution. *Outliers* are indicated by small circles in the plot.

bits per string, recombination probability to 0.9, binary tournament selection method was used, and a standard uniform crossover operator was chosen.

The results are shown in Figure 1.5, were the relative distances to the best solution found by any of the algorithms are shown. As it can be seen, the hybrid algorithm outperforms the original algorithms in most cases.

Figure 1.6 shows the on-line evolution of the lower bound for the three algorithms on two instances. Notice how the hybrid algorithm yields consistently better results all over the run. This confirms the goodness of the hybrid model as an anytime algorithm.

1.4.2 Results for second model

We solved the same problems using the EA, a Beam Search (BS) algorithm ($k_{bw} = 100$)–, and the second hybrid algorithm. The upper bound was obtained solving the LP-relaxation of the problem in all cases. A single execution for each instance was performed for the BS method whereas ten independent runs per instance were carried out for the EA and hybrid algorithms. The algorithms were run for 600 seconds in all cases. For the EA and the hybrid algorithm, the size of the population was fixed at 100 individuals that were initialized with random feasible solutions. With the aim of maintaining some diversity in the population, duplicated individuals were not allowed in

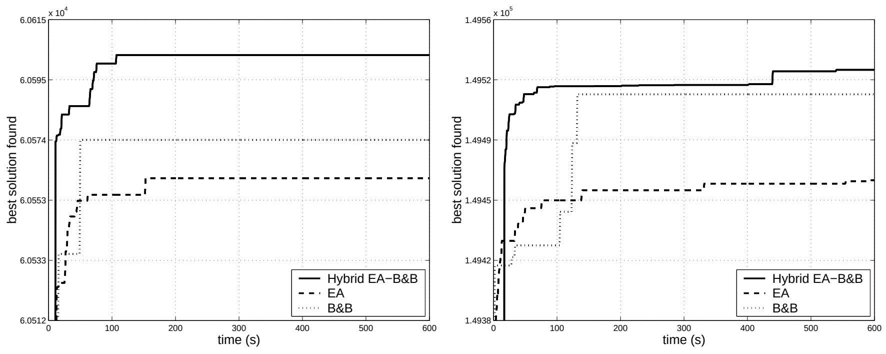

Fig. 1.6 Temporal evolution of the lower bound in the three algorithms for an problem instance with $\alpha = .75$, $m = 30$, $n = 100$ (Left) and $\alpha = .75$, $m = 30$, $n = 250$ (Right). Curves are averaged for the ten runs in the case of the EA and the hybrid algorithm.

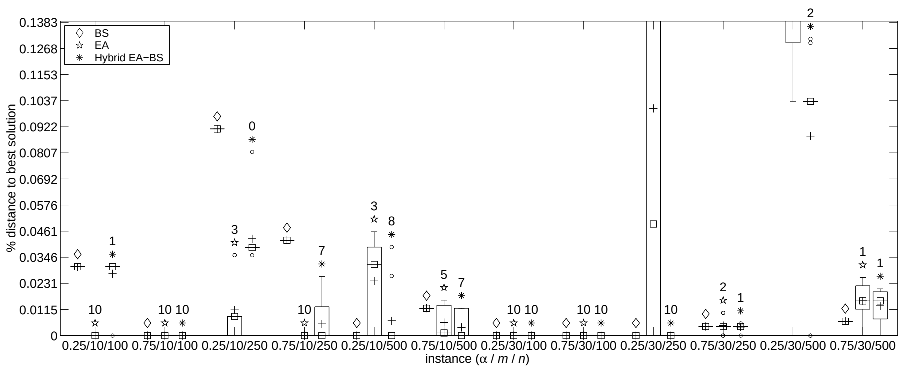

Fig. 1.7 Results of the BS algorithm, the EA, and the second hybrid model for problem instances of different number of items (n), knapsacks (m), and tightness ratio (α).

the population. The crossover probability was set to 0.9, binary tournament selection was used, and standard uniform crossover operator was chosen.

Execution results for BS algorithm, the EA and the hybrid model are shown in Figure 1.7. As it can be seen, the hybrid algorithm provides better results for the largest problem instances regardless of the tightness ratio. For the smallest problem instances, the EA performs better. This may be due to the lower difficulty of the latter instances; the search overhead of switching from the EA to the B&B may not be worth in this case. The hybrid algorithm just starts being advantageous in larger instances, where the EA faces a more

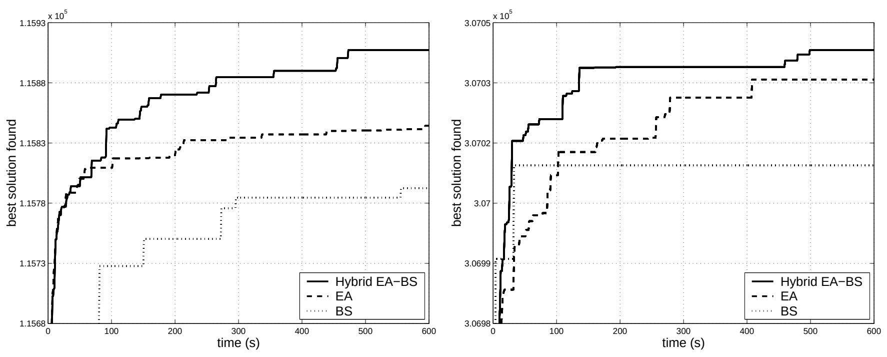

Fig. 1.8 Evolution of the best solution in the evolutionary algorithm (EA), beam search, and the hybrid algorithm during 600 seconds of execution. (Left) problem instance with $\alpha = .25$, $m = 30$, $n = 500$. (Right) problem instance with $\alpha = .75$, $m = 10$, $n = 500$. In both cases, curves are averaged for ten runs for the EA and the hybrid algorithm.

difficult optimization scenario. Notice also that the hybrid algorithm is always able to provide a solution better or equal to the one provided by BS.

Figure 1.8 shows the evolution of the best value found by the different algorithms for two specific problem instances. Note that the hybrid algorithm always provides here better results than the original ones, specially in the case of the more constrained instance ($\alpha = .25$).

1.5 CONCLUSIONS AND FUTURE WORK

We have presented hybridizations of an EA with a B&B algorithm. The EA provides lower bounds that the B&B can use to purge the problem queue, whereas the B&B guides the EA to look into promising regions of the search space. The resulting hybrid algorithm has been tested on large instances of the MKP problem with encouraging results: the hybrid EA produces better results than the constituent algorithms at the same computational cost. This indicates the synergy of this combination, thus supporting the idea that this is a profitable approach for tackling difficult combinatorial problems.

Acknowledgments

The authors are partially supported by the Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). First and third authors are also supported under contract TIN-2007-67134.

References

1. E. L. Lawler and D. E. Wood. Branch and bounds methods: A survey. *Operations Research*, 4(4):669–719, 1966.

2. T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York NY, 1996.

3. T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.

4. A. E. Eiben and J. E. Smith. *Introduction to evolutionary computation.* Springer-Verlag, 2003.

5. L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York NY, 1991.

6. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

7. J. Culberson. On the futility of blind search: An algorithmic view of "no free lunch". *Evolutionary Computation*, 6(2):109–128, 1998.

8. P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, Maidenhead, Berkshire, England, UK, 1999.

9. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Press, Boston, Massachusetts, USA, 2003.

10. P. Moscato, A. Mendes, and C. Cotta. Memetic algorithms. In G.C. Onwubolu and B.V. Babu, editors, *New Optimization Techniques in Engineering*, pages 53–85. Springer-Verlag, Berlin Heidelberg, 2004.

11. N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

12. A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

13. A. Barr and E. A. Feigenbaum. *Handbook of Artificial Intelligence.* Morgan Kaufmann, New York NY, 1981.

14. P. Wiston. *Artificial Intelligence.* Addison-Wesley, Reading MA, 1984.

15. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

16. D. R. Anderson, D. J. Sweeney, and T. A. Willians. *Introduction to Managment Science: Quantitative Approaches to Decision Making.* West Publishing, 1997.

17. G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation.* Wiley, 1951.

18. N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

19. W. E. Hart and R. K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195, San Mateo CA, 1991. Morgan Kaufmann.

20. W. E. Hart, N. Krasnogor, and J. E. Smith. *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing.* Springer-Verlag, Berlin Heidelberg, 2005.

21. R. Dawkins. *The Selfish Gene.* Clarendon Press, Oxford, 1976.

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

23. P. Moscato and C. Cotta. Memetic algorithms. In T. F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics.* Chapman & Hall/CRC, 2007. Chapter 27.

24. H. Salkin and K. Mathur. *Foundations of Integer Programming.* North Holland, 1989.

25. O. H. Ibarra and C. E. Kim. Fast approximation for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

26. B. Korte and R. Schrader. On the existence of fast approximation schemes. In O. L. Mangasarian, R. R. Meyer, and S. Robinson, editors, *Nonlinear Programming 4*, pages 415–437. Academic Press, 1981.

27. A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.

28. D. Lichtenberger. An extended local branching framework and its application to the multidimensional knapsack problem. Diploma thesis, Institut für Computergrafik und Algorithmen, Technischen Universität Wien, 2005.

29. E. Balas and H. Martin. Pivot and Complement - a heuristic for 0-1 programming. *Management Science*, 26(1):86–96, 1980.

30. C. Cotta and J. M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 3*, pages 251–255, Wien New York, 1998. Springer-Verlag.

31. J. Gottlieb. Permutation-based evolutionary algorithms for multidimensional knapsack problems. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 408–414. ACM Press, 2000.

32. S. Khuri, T. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, editors, *Proceedings of the 1994 ACM Symposium on Applied Computation*, pages 188–193. ACM Press, 1994.

33. G. R. Raidl. An improved genetic algorithm for the multiconstraint knapsack problem. In *Proceedings of the 5th IEEE International Conference on Evolutionary Computation*, pages 207–211, 1998.

34. G. R. Raidl and J. Gottlieb. Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. Technical Report TR 186–1–04–05, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2004.

35. H. Ishibuchi, S. Kaige, and K. Narukawa. Comparison between Lamarckian and Baldwinian repair on multiobjective 0/1 knapsack problems. In C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005*, volume 3410 of *Lecture Notes in Computer Science*, pages 370–385, Berlin Heidelberg, 2005.

36. I. Dumitrescu and T. Stutzle. Combinations of local search and exact algorithms. In G.L. Raidl et al, editor, *Applications of evolutionary computation*, volume 2611 of *Lecture Notes in Computer Science*, pages 211–223. Springer-Verlag, Berlin, 2003.

37. S. Fernandes and H. Lourenço. Hybrid combining local search heuristics with exact algorithms. In F. Almeida et al, editor, *V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 269–274, 2007.

38. J. Puchinger and G. R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification. In J. Mira and J. R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 41–53, Berlin Heidelberg, 2005. Springer-Verlag.

39. M. Vasquez and J. K. Hao. A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 328–333, 2001.

40. J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Proceedings of the 6th IEEE International Conference on Evolutionary Computation*, pages 2317–2324. IEEE Press, 1999.

41. C. Cotta and J. M. Troya. Embedding branch and bound within evolutionary algorithms. *Applied Intelligence*, 18(2):137–153, 2003.

42. C. Cotta, J. F. Aldana, A. J. Nebro, and J. M. Troya. Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In D.W. Pearson, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 2*, pages 277–280, Wien New York, 1995. Springer-Verlag.

43. J. Puchinger, G. R. Raidl, and G. Koller. Solving a real-world glass cutting problem. In J. Gottlieb and G. R. Raidl, editors, *4th European Conference on Evolutionary Computation in Combinatorial Optimization*, volume 3004 of *Lecture Notes in Computer Science*, pages 165–176, 2004.

44. A. P. French, A. C. Robinson, and J. M. Wilson. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7(6):551–564, 2001.

45. K. Kostikas and C. Fragakis. Genetic programming applied to mixed integer programming. In M. Keijzer, U. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, editors, *7th European Conference on Genetic Programming*, volume 3003 of *Lecture Notes in Computer Science*, pages 113–124, Coimbra, Portugal, 2004. Springer-Verlag.

46. A. Volgenant and R. Jonker. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research*, 9:83–88, 1982.

47. J. E. Gallardo, C. Cotta, and A. J. Fernández. Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In J. Mira and J. R. Álvarez, editors, *Proceedings of the 1st International Work-conference on the Interplay between Natural and Artificial Computation*, volume 3562 of *Lecture Notes in Computer Science*, Berlin Heidelberg, 2005. Springer-Verlag.

48. J. E. Gallardo, C. Cotta, and A. J. Fernández. A hybrid model of evolutionary algorithms and branch-and-bound for combinatorial optimization problems. In *2005 Congress on Evolutionary Computation*, pages 2248–2254, Edinburgh, UK, 2005. IEEE Press.

49. J. E. Gallardo, C. Cotta, and A. J. Fernández. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man and Cybernetics, part B*, 37(1):77–83, 2007.

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