Solving the Multidimensional Knapsack Problem Using an Evolutionary Algorithm Hybridized with Branch and Bound

José E. Gallardo, Carlos Cotta, and Antonio J. Fernández

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, Spain
{pepeg, ccottap, afdez}@lcc.uma.es

Abstract. A hybridization of an evolutionary algorithm (EA) with the branch and bound method (B&B) is presented in this paper. Both techniques cooperate by exchanging information, namely lower bounds in the case of the EA, and partial promising solutions in the case of the B&B. The multidimensional knapsack problem has been chosen as a benchmark. To be precise, the algorithms have been tested on large problems instances from the OR-library. As it will be shown, the hybrid approach can provide high quality results, better than those obtained by the EA and the B&B on their own.

1 Introduction

Branch and Bound (B&B) [1] is an algorithm for finding optimal solutions to combinatorial problems. Basically, the method produces convergent lower and upper bounds for the optimal solution using an implicit enumeration scheme. The algorithm starts from the original problem, and proceeds iteratively. In each stage, the problem is split into subproblems such that the union of feasible solutions for these subproblems gives the whole set of feasible solutions for the current problem. Subproblems are further divided until they are solved, or their upper bounds are below the best feasible solution found so far (maximization is assumed here). Thus, the approach produces a branching tree in which each node corresponds to a problem and the children of the node represent the subproblems into which it is split. Several strategies can be used to traverse the search tree. The most efficient one consists of expanding more promising (according to the attainable solution, i.e., the upper bound) problems first, but memory resources may be exhausted. A depth-first expansion requires less memory, but will likely expand much more nodes than the previous strategy.

A different approach to optimization is provided by evolutionary algorithms [2, 3] (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 [4, 5, 6]. A promising approach to achieve this knowledge-augmentation is the hybridization with other (domain-specific) heuristics for the optimization problem to be solved. In this paper a hybridization of an EA with B&B is presented. This hybridization is aimed to combining their search capabilities in a synergetic way.

The remainder of the article is organized as follows: Sect. 2 presents the *multidimensional knapsack problem* (MKP) –the benchmark used to test our hybrid model– and describes both an efficient evolutionary algorithm and two different B&B implementations that have been successfully applied to solve the MKP. Then, Sect. 3 discusses related work regarding the hybridization of evolutionary algorithms and B&B models; a novel proposal for this hybridization is described here too. Subsequently, Sect. 4 shows and analyzes the empirical results obtained by the application of each of the described approaches (i.e., the EA, pure B&B models and the hybrid model) on different instances of the benchmark. Finally, Sect. 5 provides the conclusions and outlines ideas for future work.

2 The Multidimensional Knapsack Problem

Let us firstly describe the target problem, and several approaches –both meta-heuristic and exact– used for solving it.

2.1 Description of the Problem

The *Multidimensional Knapsack Problem* (MKP) is a generalization of the classical *knapsack problem*, so it is worth starting with the description of the latter. 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. Formally, the problem can be formulated as:

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

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3}$$

Each of the m constraints in Eq. (2) is called a knapsack constraint, and vector $\boldsymbol{x}$ describes which objects are chosen in the solution. The problem is NP-hard [7], and 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. [8]).

2.2 An Evolutionary Algorithm

EAs have been used in several works for solving the MKP, e.g., [9, 10, 11, 12, 13] among others. To the best of our knowledge, the EA developed by Chu and Beasley in [11] represents the state-of-the-art in solving the MKP with EAs. This particular algorithm has an additional advantage: it uses the *natural* representation of solutions, i.e., n-bit binary strings, where n is the number of items in the MKP. For this representation, a value of 0 or 1 in the j-th bit indicates the value of x_j in the solution.

Since this representation allows infeasible solutions, a repair operator is used to correct them. In order to implement this operator, a preprocessing routine is first applied to each problem to sort variables according to the decreasing order of their *pseudo-utility* ratios u_j's (the greater the ratio, the higher the chance that the corresponding variable will be set to one in the solution, see [11] for details). Then, an algorithm consisting in two phases (see Fig. 1) is applied to every solution. In the first phase, variables are examined in increasing order of u_j and set to zero if feasibility is violated. In the second phase, variables are examined in reverse order and set to one as long as feasibility is not violated. The aim of the first phase is to obtain a feasible solution, whereas the second phase seeks to improve its fitness.

By restricting the EA to search only the feasible region of the solution space, the simple fitness function $f(\boldsymbol{x}) = \sum_{j=1}^{n} p_j x_j$ can be considered.

```
1:  initialize $R_i = \sum_{j=1}^{n} r_{ij} x_j, \forall i \in \{1, \cdots, n\}$;
2:  **for** j = n **down to** 1 **do**      /* *DROP phase* */
3:      **if** $(x_j = 1)$ and $(\exists i \in \{1, \cdots, n\} : R_i > b_i)$ **then**
4:          set $x_j \leftarrow 0$;
5:          set $R_i \leftarrow R_i - r_{ij}, \forall i \in \{1, \cdots, n\}$
6:      **end if**
7:  **end for**
8:  **for** j = 1 **up to** n **do**      /* *ADD phase* */
9:      **if** $(x_j = 0)$ and $(\forall i \in \{1, \cdots, n\} : R_i + r_{ij} \le b_i)$ **then**
10:         set $x_j \leftarrow 1$;
11:         set $R_i \leftarrow R_i + r_{ij}, \forall i \in \{1, \cdots, n\}$
12:     **end if**
13: **end for**
```

Fig. 1. Repair operator for the MKP

2.3 Branch and Bound Algorithms

Two B&B algorithms have been evaluated. The first one is a simple implementation that expands the search tree by introducing or excluding an arbitrary item in the knapsack until a complete solution is generated. When an item j is included, the lower bound for the problem 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. Of course, infeasible solutions are pruned during the process. The problem queue is examined in a depth-first way in order to avoid memory exhaustion when solving large problems. Although this method is very naive, it can be very efficiently implemented and may be the only one available for other problems for which no sophisticated heuristics have been developed.

The second implementation performs a standard linear programming (LP)-based tree search. The algorithm solves the linear relaxation of the current problem (that is, allowing fractional values for decision variables), and considers the problem solved if all variables have integral values in the relaxed solution. Otherwise, it expands two new problems by introducing or excluding an item with an associated fractional value (the one whose value in the LP-relaxed solution is closest to 0.5). This method is more accurate, but is not very fast when large problems are considered, as their relaxation may take some time to be solved.

3 Hybrid Models

In this section we present a hybrid model that integrates an EA 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 B&B techniques and EAs.

3.1 Related Work

Cotta *et al.* [14] used a problem-specific B&B approach for the traveling salesman problem based on 1-trees and the Lagrangean relaxation [15], and made use of an EA to provide bounds in order to guide the B&B search. More specifically, they analyzed two different approaches for the integration. In the first model, the genetic algorithm plays the role of master and the B&B is incorporated as a tool of it. The primary idea was to build a hybrid genetic operator based in the B&B philosophy. The second model proposed consisted of executing in parallel the B&B algorithm with a certain number of EAs which generate a number of solutions of different structure. 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 [14] described only preliminary results.

Another relevant research was developed by Nagard *et al.* [16], combining a B&B tree search and an EA which was used to provide bounds for solving flowshop scheduling problems. Later, a hybrid algorithm, combining genetic algorithms and integer programming B&B approaches to solve MAX-SAT problems was described by French *et al.* [17]. This hybrid algorithm gathers information during the run of a linear programming based B&B algorithm, and uses it to build an EA population. The EA is eventually activated, and the best solution found is used to inject new nodes in the B&B search tree. The hybrid algorithm is run until the search tree is exhausted, and hence it is an exact approach. However, in some cases it can expand more nodes than the B&B alone.

More recently, Cotta and Troya [18] presented a framework for the hybridization based on using B&B as an 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 metaheuristic provides better results than pure EAs in problems where a full B&B exploration is unpractical on its own.

3.2 Our Hybrid Algorithm

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 (see Fig. 2), 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.

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

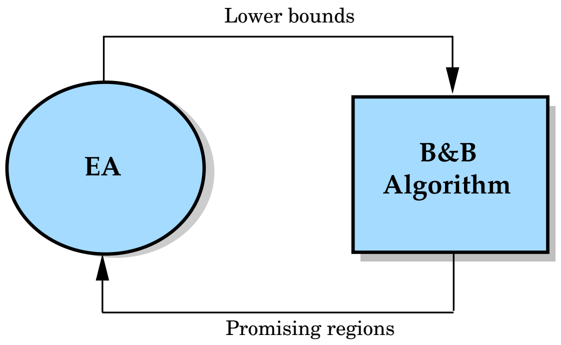

Fig. 2. The hybrid algorithm

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.

4 Experimental Results

We have tested our algorithms with problems available at the OR-library [19] 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.

We solved these problems on a Pentium IV PC (1700MHz and 256MB of main memory) using the EA, the B&B and the hybrid algorithms (all of them coded in C). A single execution for each instance was performed for the B&B method 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, the size of the population was fixed at 100 individuals that were initialized with random feasible solutions. The probability of mutation was set to 2 bits per string, recombination probability was set to 0.9, the binary tournament selection method was used, and a standard uniform crossover operator was chosen.

Table 1. Results (averaged for ten runs) of the B&B algorithm, the EA, and the hybrid thereof for problem instances of different number of items (n), knapsacks (m), and tightness ratio (α)

				GA		B&B-GA	
α	m	n	B&B	best	mean ± std.dev	best	mean ± std.dev
0.25	5	100	24373	24381	24381.0 ± 0.0	24381	24381.0 ± 0.0
		250	59243	59243	59211.7 ± 18.0	59312	59305.1 ± 20.7
		500	120082	120095	120054.0 ± 25.1	120148	120122.0 ± 14.0
	10	100	23064	23064	23050.2 ± 19.2	23064	23059.1 ± 3.2
		250	59071	59133	59068.7 ± 29.1	59164	59146.3 ± 11.6
		500	117632	117711	117627.3 ± 64.7	117741	117702.4 ± 20.5
	30	100	21516	21946	21856.1 ± 112.5	21946	21946.0 ± 0.0
		250	56277	56796	56606.9 ± 126.6	56796	56796.0 ± 0.0
		500	115154	115763	115619.9 ± 79.7	115820	115779.6 ± 18.6
0.75	5	100	59960	59960	59960.0 ± 0.0	59965	59965.0 ± 0.0
		250	154654	154668	154626.2 ± 31.7	154668	154668.0 ± 0.0
		500	299904	299885	299842.7 ± 26.9	299904	299902.3 ± 5.1
	10	100	60633	60633	60629.7 ± 4.9	60633	60633.0 ± 0.0
		250	149641	149686	149622.7 ± 39.6	149704	149685.3 ± 15.1
		500	306949	306976	306893.7 ± 56.0	307027	307002.7 ± 8.4
	30	100	60574	60593	60560.9 ± 32.1	60603	60603.0 ± 0.0
		250	149514	149514	149462.8 ± 44.4	149595	149528.6 ± 24.4
		500	300309	300351	300218.8 ± 94.5	300387	300359.0 ± 21.9

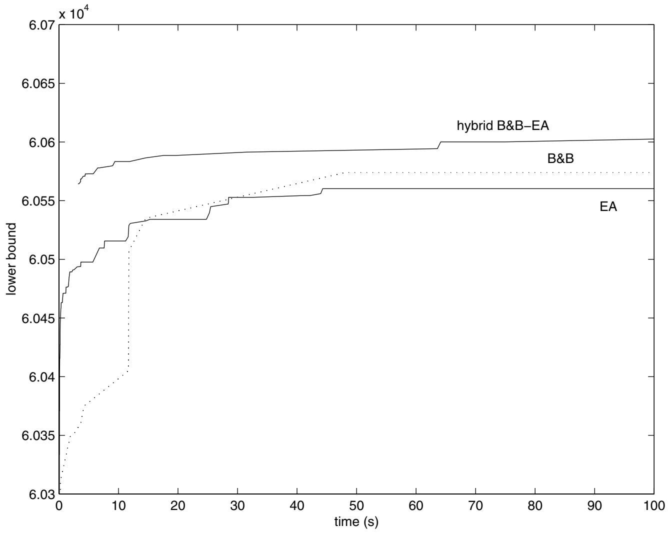

Fig. 3. Evolution of the lower bound in the three algorithms during the first 100 seconds of execution for an problem instance with $\alpha = .75$, $m = 30$, $n = 100$. Curves are averaged for the ten runs in the case of the EA and the hybrid algorithm

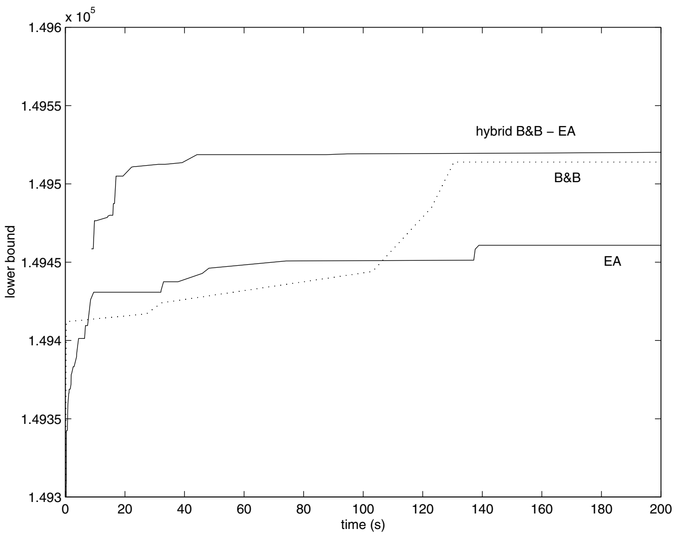

Fig. 4. Evolution of the lower bound in the three algorithms during the first 100 seconds of execution for an problem instance with $\alpha = .75$, $m = 30$, $n = 250$. Curves are averaged for the ten runs in the case of the EA and the hybrid algorithm

The results are shown in Table 1. The first three columns indicate the sizes (m and n) and the tightness ratio (α) for a particular instance. The next column reports results for the B&B algorithm, whereas the last two columns report the best and average solutions over 10 runs for the EA and the hybrid algorithm. As it can be seen, the hybrid algorithm always outperforms the original algorithms. Notice also that the difference in the mean values is notably larger than the corresponding standard deviations, thus reinforcing the significance of the results.

Figs. 3 and 4 show the on-line evolution of the lower bound for the three algorithms. 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.

We are currently testing the second implementation of the hybrid algorithm that solves LP-relaxation of the problems. The preliminary results indicate that the lower bounds obtained by this algorithm are not better than the ones reported in this paper, although more accurate upper bounds can be achieved.

5 Conclusions and Future Work

We have presented a hybridization 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. In this sense, further work will be directed to confirm these findings on different combinatorial problems, as well as to study alternative models for the hybridization of the B&B with EAs.

Acknowledgements

This work is partially supported by Spanish MCyT and FEDER under contracts TIC2002-04498-C05-02 and TIN2004-7943-C04-01.

References

1. Lawler, E., Wood, D.: Branch and bounds methods: A survey. Operations Research **4** (1966) 669–719
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York NY (1996)
3. Bäck, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation. Oxford University Press, New York NY (1997)
4. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York NY (1991)
5. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1** (1997) 67–82
6. Culberson, J.: On the futility of blind search: An algorithmic view of "no free lunch". Evolutionary Computation **6** (1998) 109–128
7. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman and Co., San Francisco CA (1979)
8. Salkin, H., Mathur, K.: Foundations of Integer Programming. North Holland (1989)
9. Khuri, S., Bäck, T., Heitkötter, J.: The zero/one multiple knapsack problem and genetic algorithms. In Deaton, E., Oppenheim, D., Urban, J., Berghel, H., eds.: Proceedings of the 1994 ACM Symposium on Applied Computation, ACM Press (1994) 188–193
10. Cotta, C., Troya, J.: A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In Smith, G., Steele, N., Albrecht, R., eds.: Artificial Neural Nets and Genetic Algorithms 3, Wien New York, Springer-Verlag (1998) 251–255
11. Chu, P., Beasley, J.: A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics **4** (1998) 63–86
12. Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In Carroll, J., Damiani, E., Haddad, H., Oppenheim, D., eds.: ACM Symposium on Applied Computing 2000, ACM Press (2000) 408–414
13. Raidl, G., Gottlieb, J.: 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)

14. Cotta, C., Aldana, J., Nebro, A., Troya, J.: Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In Pearson, D., Steele, N., Albrecht, R., eds.: Artificial Neural Nets and Genetic Algorithms 2, Wien New York, Springer-verlag (1995) 277–280
15. Volgenant, A., Jonker, R.: A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. European Journal of Operational Research **9** (1982) 83–88
16. Nagard, A., Heragu, S., Haddock, J.: A combined branch and bound and genetic algorithm based for a flowshop scheduling algorithm. Annals of Operation Research **63** (1996) 397–414
17. French, A., Robinson, A., Wilson, J.: Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. Journal of Heuristics **7** (2001) 551–564
18. Cotta, C., Troya, J.: Embedding branch and bound within evolutionary algorithms. Applied Intelligence **18** (2003) 137–153
19. Beasley, J.: Or-library: distributing test problems by electronic mail. Journal of the Operational Research Society **41** (1990) 1069–1072