On the Hybridization of Memetic Algorithms with Branch-and-Bound Techniques

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

Abstract

Branch-and-Bound and memetic algorithms represent two very different approaches for tackling combinatorial optimization problems. These approaches are not incompatible however. In this paper, we consider a hybrid model that combines these two techniques. To be precise, it is based on the interleaved execution of both approaches. Since the requirements of time and memory in branch-and-bound techniques are generally conflicting, we have opted for carrying out a truncated exact search, namely, beam search. The resulting hybrid algorithm has therefore a heuristic nature. The multidimensional 0-1 knapsack problem and the shortest common supersequence problem have been chosen as benchmarks. As will be shown, the hybrid algorithm can produce better results in both problems at the same computational cost, specially for large problem instances.

I. Introduction

Branch-and-bound techniques (BnB) [1] constitute a well-known approach for solving combinatorial optimization problems to optimality. Essentially, BnB techniques use an implicit enumeration scheme for exploring the search space in an *intelligent* way. This is done by partitioning the search space, producing upper and lower bounds of the solutions attainable in each partition. Thus, the search performed by the algorithm can be represented as a tree traversed in a certain way. The most efficient (in terms of the number of iterations required to find the optimum and prove its optimality) is best-first, i.e., expanding firstly the most promising –according to the local bound– nodes. However, the memory requirements can make this strategy unrealistic for large problem instances. The alternative is to use a depth-first traversal. This strategy does not require large amounts of memory, but it can expand many more nodes than best-first. Besides these simple strategies, leading BnB software also uses more sophisticated selection procedures such as *best first with diving* (i.e., a mixture of depth-first and best-first) to quickly obtain good incumbent solutions. For an in-depth discussion of search strategies for mixed-integer programming see [2].

On the other hand, evolutionary algorithms (EAs) [3] have a completely different philosophy: tentative solutions are iteratively generated, aiming at producing better and better solutions. Their performance is *probably*, yet not *provably*, good: near-optimal solutions can be typically found at an acceptable computational cost in many

The three authors are with the Department *Lenguajes y Ciencias de la Computación*, Málaga University, Spain, {pepeg,ccottap,afdez}@lcc.uma.es. This work was partially supported by Spanish MCyT and FEDER under contracts TIN2004-7943-C04-01 and TIN2005-08818-C04-01. We thank the reviewers for their useful comments and suggestions.

combinatorial optimization problems. It should be noted that –despite the underlying algorithmic template of EAs being pretty much the same in these different problems– the need to exploit problem-knowledge has been repeatedly shown in theory and in practice [4]. Different attempts have been made to answer this need; memetic algorithms [5] (MAs) is probably one of the most successful to date [6].

MAs [7] were originally conceived as a family of metaheuristics that tried to blend several concepts from tightly separated –at that time– families such as EAs and simulated annealing. In essence, a MA is a search strategy in which a population of optimizing agents synergistically cooperate and compete. This behavior can be accomplished by using local-search strategies within a population-based search technique such as an EA, although it must be noted that the MA paradigm does not simply reduce itself to this particular scheme.

Regarding this matter, we present here a model for further hybridizing MAs (and EAs in general) with BnB techniques. The goal is to synergistically combine these two different solving approaches, exploiting the capability of BnB for identifying provably good regions of the search space, and the potential of MAs for exploring these. The next section will describe the model that we have used for this purpose. Subsequently, we will proceed to the deployment of this hybrid approach on two combinatorial problems: the multidimensional 0-1 knapsack problem (MKP) and the shortest common supersequence problem (SCSP).

II. The Hybrid Model

Despite the complementarity of exact techniques and metaheuristics (in particular EAs), combined approaches are not plentiful in the literature. The reader is referred to [8] for an overview of these approaches. In general, hybrid methods can be classified as *coercive* (one technique plays the role of master, and uses the other as a subordinate tool) and *cooperative* (both techniques work at the same level, and aim for a symbiotic collaboration). In this work, we have considered this last approach in order to integrate evolutionary techniques and BnB models. This is done by letting both techniques work in an intertwined way (i.e., both processes are allowed to perform independently). By doing so, both processes will share the incumbent solution so that the following benefits can be obtained:

- The BnB can use the current solution to purge the problem queue, deleting those problems whose local bound is smaller than the one obtained by the MA.
- The BnB can inject information about more promising regions of the search space into the MA population in order to guide the memetic search.

In order to explore the search space, BnB methods can traverse the tree in different ways as mentioned in Section I. 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 (according to some *quality* meassure) k nodes. This implies sacrificing exactness, but provides a very effective heuristic search

approach. The name *beam search* (BS) has been coined to denote this strategy [9]. An effective hybrid model can be obtained by integrating this latter approach with a MA as shown in the following pseudocode:

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

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 best nodes in the BS queue. 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 a problem dependent way. The intended goal of this initialization is to lead the MA search to these regions of the search space (recall that the nodes in the queue represent subsets of the search space considered promising by the BnB; hence, the MA is used for finding probably good solutions in this region). Upon stabilization of the MA, control is returned to the BnB algorithm. The lower bound for the optimal solution obtained by the MA is then compared to the current incumbent in the BnB, 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.

III. Materials and Methods

In this section, we introduce two difficult combinatorial problems that we have used to evaluate our model, namely the MKP (a maximization problem) and the SCSP (a minimization problem).

A. The Multidimensional 0-1 Knapsack Problem

The MKP is a generalization of the classical 0-1 knapsack problem, so it is of benefit to first describe this problem. An instance of the classical 0-1 knapsack problem is defined by a knapsack of capacity b, and a set of n objects $O = \{o_1, \cdots, o_n\}$. Each of these objects o_j has a value p_j and a weight r_j. The problem amounts to selecting a subset $S \subseteq O$ of objects, such that their combined weight does not exceed the knapsack capacity, and their value is maximal.

The MKP generalizes the previous definition by considering m different knapsacks, each of them with a possibly different capacity b_i. The subset of objects selected must fit simultaneously within all m knapsacks. Furthermore,

objects have a different weight r_{ij} within each knapsack. This way, the problem can be formalized as follows:

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

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

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

$$\text{where} \qquad p_j > 0, \quad r_{ij} \geqslant 0, \quad b_i \geqslant 0 \tag{4}$$

Thus, vector $\vec{x}$ describes the objects selected in the solution. The problem can be regarded as a general statement of binary integer programming with non-negative coefficients. Many real-world problems can be formulated like the MKP, e.g., capital budgeting, project selection, etc. As regards this, it must be noted that the MKP has been shown to be NP$-$hard; furthermore, it is known that it admits no fully polynomial time approximation scheme (FPTAS) (unless P=NP) [10].

1) A Branch-and-Bound Approach to the MKP: The BnB algorithm considered carries out a standard exploration of the search tree for this kind of problem (see [11]), namely using the linear relaxation approach. This means assuming variables can take fractional values in the interval [0,1], and using linear-programming (LP) techniques to provide upper bounds. More precisely, each node is defined by a set of constraints $\mathcal{C} = \{C_1, \cdots, C_k\}$, where $C_k = (i_k, v_k)$ implies that x_{i_k} is forced to take the value v_k. 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 $1/2$ 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. With the aim of obtaining good incumbent solutions as soon as possible, a greedy first-fit heuristic based on the order determined by *pseudo-utility* ratios (see next subsection) is used on each node to complete partial solutions.

2) A Memetic Algorithm for the MKP: The MKP has been tackled via EAs in many works, e.g., [12]–[14]. Among these, the EA developed by Chu and Beasley [12] 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., $(x_j = 1) \Leftrightarrow o_j \in S$). Since infeasible solutions might be represented in this way, a Lamarckian repairing mechanism is used (see [15] 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 [12] 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 $f(\vec{x}) = \sum_{j=1}^{n} p_j x_j$.

B. The Shortest Common Supersequence Problem

Let us firstly introduce some notation. We write $|s|$ for the length of string s ($|s_1 s_2 \dots s_n| = n$) and ϵ for the empty string ($|\epsilon| = 0$). Abusing the notation, $|\Sigma|$ denotes the cardinality of set Σ. We use $s \trianglerighteq \alpha$ for the total number of occurrences of symbol α in string s ($s_1 s_2 \dots s_n \trianglerighteq \alpha = \sum_{1 \leq i \leq n, s_i = \alpha} 1$). We write αs for the string obtained by appending the symbol α in front of string s. Finally, $s \in \Sigma^*$ means that s is a finite length string of symbols in Σ.

Let s and r be two strings of symbols taken from an alphabet Σ. String s can be said to be a supersequence of r (denoted as $s \succ r$) using the following recursive definition:

$$\begin{array}{rcll} s \succ \epsilon & \triangleq & \text{True} & \\ \epsilon \succ r & \triangleq & \text{False}, & \text{if } r \neq \epsilon \\ \alpha s \succ \alpha r & \triangleq & s \succ r & \\ \alpha s \succ \beta r & \triangleq & s \succ \beta r, & \text{if } \alpha \neq \beta \end{array} \tag{5}$$

Plainly, $s \succ r$ implies that r can be embedded in s, meaning that all symbols in r are present in s in the very same order (although not necessarily consecutive). For example, given the alphabet $\Sigma = \{a, b, c\}$, $aacab \succ acb$.

We can now state the SCSP as follows: an instance $I = (\Sigma, L)$ for the SCSP is given by a finite alphabet Σ and a set L of m strings $\{s_1, \cdots, s_m\}$, $s_i \in \Sigma^*$. The problem consists of finding a string s of minimal length that is a supersequence of each string in L (for example, given $I = (\{a, b, c\}, \{cba, abba, abc\})$, a shortest common supersequence of I is $abcba$). The SCSP can be shown to be NP$-$hard, even if strong constraints are posed on L, or on Σ (e.g., see [16]). It is also not fixed parameter tractable under several parameterizations, e.g., [17].

1) A Branch-and-Bound Approach to the SCSP: First of all, let us define the following function that will be useful to estimate lower bounds for nodes explored by this algorithm:

$$\begin{array}{rcll} s \gg \epsilon & \triangleq & (\epsilon, \epsilon) & \\ \epsilon \gg r & \triangleq & (\epsilon, r), & \text{if } r \neq \epsilon \\ \alpha s \gg \alpha r & \triangleq & (\alpha r^e, r^r), & \text{where } (r^e, r^r) = s \gg r \\ \alpha s \gg \beta r & \triangleq & s \gg \beta r, & \text{if } \alpha \neq \beta \end{array} \tag{6}$$

Intuitively, $s \gg r = (r^e, r^r)$ if r^e is the longest initial segment of r embedded by s, and r^r is the remaining part of r not embedded by s (for example, $aabbacb \gg abca = (abc, a)$). Note that $r = r^e r^r$, and $s \succ r \iff s \gg r = (r, \epsilon)$.

BnB algorithms for an instance $I = (\Sigma, L)$ of the SCSP start from a single node containing as tentative solution ϵ. Each node is then split into $|\Sigma|$ subproblems, each of them obtained by appending a symbol from Σ to the

TABLE I: Results of the different algorithms for the MKP.

			BS		MA			Hybrid MA-BS		
α	m	n	solution	time	best	mean $\pm$ σ	time	best	mean $\pm$ σ	time
0.25	10	100	23057	1.7	23064	23064.0 $\pm$ 0.0	31.5	*23064*	*23064.0* $\pm$ 0.0	98.9
		250	59133	6.1	59187	59182.5 $\pm$ 7.0	139.0	*59187*	**59183.2** $\pm$ 7.4	73.2
		500	117772	50.5	117772	117736.9 $\pm$ 20.9	249.7	**117809**	**117754.1** $\pm$ 17.8	248.9
	30	100	21946	0.8	21946	21946.0 $\pm$ 0.0	1.0	*21946*	*21946.0* $\pm$ 0.0	0.8
		250	56824	10.3	56824	56730.1 $\pm$ 54.7	209.3	*56824*	**56824.0** $\pm$ 0.0	277.0
		500	115796	588.6	115903	115856.0 $\pm$ 25.7	254.5	**116014**	**115893.2** $\pm$ 30.6	240.1
0.75	10	100	60633	3.7	60633	60633.0 $\pm$ 0.0	0.9	*60633*	*60633.0* $\pm$ 0.0	0.7
		250	149641	0.3	149704	149703.2 $\pm$ 3.7	117.0	*149704*	**149704.0** $\pm$ 0.0	59.3
		500	307013	24.1	307050	307040.2 $\pm$ 14.3	199.1	**307072**	**307051.8** $\pm$ 12.9	430.1
	30	100	60603	0.2	60603	60603.0 $\pm$ 0.0	2.7	*60603*	*60603.0* $\pm$ 0.0	3.4
		250	149595	32.6	**149601**	149585.5 $\pm$ 16.8	209.5	149595	**149592.4** $\pm$ 8.8	510.0
		500	300512	124.5	300512	300463.7 $\pm$ 23.9	169.4	**300531**	**300471.9** $\pm$ 27.5	250.5

current tentative string. Nodes with unproductive characters (i.e., not contributing to embedding any string in L) are pruned from the search tree.

To obtain a lower bound for a node with tentative solution s^t, the set of remaining strings in L not embedded by s^t must first be calculated as follows: let $R = \{r_i \mid (s_i^e, r_i) = s^t \gg s_i, s_i \in L\}$. Let $M(\alpha, R) = \max\{r_i \trianglerighteq \alpha \mid r_i \in R\}$ be the maximum number of occurrences of symbol α in any string in R. Clearly, every common supersequence for the remaining strings must contain at least $M(\alpha, R)$ copies of the symbol α. Thus a lower bound is given by $|s^t| + \sum_{\alpha \in \Sigma} M(\alpha, R)$, that is the length of the tentative solution plus the maximum number of occurrences in any string in R of each symbol of the alphabet. Similarly, a trivial supersequence can be calculated by concatenating the remaining strings to the tentative solution. Hence, $|s^t| + \sum_{r_i \in R} |r_i|$ serves as an upper bound.

The WMM heuristic (see next section) can be further used on each node with the aim of improving the incumbent solution. At any rate, Fraser [18] has shown that BnB algorithms alone need too much time to be practical, except for very small alphabets. As we will show, a hybrid algorithm based on BnB and a MA can provide better performance.

2) Heuristics for the SCSP: The hardness results mentioned previously motivate the utilization of heuristic approaches for tackling the SCSP. One of the most popular algorithms for this purpose is MAJORITY MERGE (MM). This is a greedy algorithm that constructs a supersequence incrementally by adding the symbol most frequently found at the front of the strings in L (ties are randomly broken), and removing these symbols from the corresponding strings.

The myopic functioning of MM makes it incapable of grasping the global structure of strings in L. In particular, MM misses the fact that the strings can have different lengths [19]. This implies that symbols at the front of short strings will have more chances of being removed, since the algorithm still has to scan the longer strings. For this reason, Branke *et al.* [19] propose to weight each occurrence of a symbol at the front of a string precisely according to the length of the string. This modified heuristic will be termed WEIGHTED MAJORITY MERGE (WMM). They

also present results from the combination of EAs and WMM. In this heuristic, the EA is used to evolve weights for each character of every string.

Another heuristic was proposed by Rahmann [20] in the context of the application of the SCSP to a microarray production setting. This algorithm is termed ALPHABET-LEFTMOST (AL), and takes as input the list of strings whose supersequence is sought, and a permutation Π of all symbols in the alphabet. The algorithm then proceeds with successive repetitions of this pattern until all the strings in L are embedded. Obviously, unproductive steps (i.e., when the next symbol in a row does not appear at the front of any string in L) are ignored.

3) A Memetic Algorithm for the SCSP: One of the difficulties faced by an EA (or by a MA) when applied to the SCSP is the existence of feasibility constraints, i.e., an arbitrary string $s \in \Sigma^*$, no matter its length, is not necessarily a supersequence of strings in L. As in the case of the MKP, this can be dealt with by penalizing, by repairing, or by defining closed operators in feasible space. We have analyzed these three approaches elsewhere [21], and we have found that repairing provided better results than penalizing or using closed operators. We will thus elaborate on the repairing option.

Our MA evolves sequences in $|\Sigma|^\lambda$, where $\lambda = \sum_{s_i \in L} |s_i|$. Before being submitted for evaluation, these sequences are repaired using a two-phase procedure: firstly, the sequence is scanned from left to right, removing unproductive steps as in AL; if the scan is completed before all strings in L have been embedded, the MM heuristic is applied to the remaining strings to complete a feasible solution.

Besides the basic improvement arising from the removal of unproductive steps, an additional local-improvement level is considered. To do so, we have considered the neighborhood defined by symbol removals, i.e., extracting a symbol from the sequence and then repairing it. A full local-search (LS) scheme is defined by iterating this operation from left to right until no single deletion results in length reduction. The improvement in solution quality attainable via the application of this LS operator comes obviously at the expense of an increased computational cost. This additional cost might be too high if LS were massively applied. On the other hand, the extreme option of simply removing LS handicaps the search capabilities of the algorithm. A pragmatic solution can be found in the use of partial Lamarckism, namely using LS at some intermediate rate.

IV. EXPERIMENTAL RESULTS

In this section we compare our hybrid model to other heuristics using both the MKP and SCSP. The experiments have been performed in all cases on a Pentium IV PC (2400MHz and 512MB RAM). Algorithms were coded in C and compiled using gcc 3.2. In all cases, a steady-state MA ($popsize = 100$, $p_X = 0.9$, $p_m = 1/n$), with binary tournament selection, uniform crossover and bit-flip mutation has been used. With the aim of maintaining diversity, duplicated individuals were not allowed in the population.

A. Experimental Results for the MKP

A hybrid algorithm for the MKP was obtained from the general description in Section II with the following parameters: $k = 100$, $l_0 = 0$, $l = 1$ (see Section IV-C for a more detailed explanation).

We tested our algorithms with problems available at the OR-library. We took two instances per problem set. Each problem instance 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 (the capacity of the i-th knapsack is $\alpha \sum_{j=1}^{n} r_{ij}$).

A single execution for each instance was performed for the BS method (since it is a deterministic method) whereas 25 independent runs per instance were carried out for the MA and hybrid algorithm. The algorithms were run for 600 seconds in all cases. For the MA and the hybrid algorithm, the population was initialized with random feasible solutions.

Execution results are shown in Table I. The first three columns indicate the tightness ratio (α) and the sizes (m and n) for a particular instance. The next columns report the best solution found and the time (in seconds) consumed to find it. Specifically for the MA and hybrid algorithm, we also show the mean of the values obtained and standard deviations, and the median time to obtain the best solution. For clarity, the best results per problem set (excluding ties) are written in bold face, and entries in italics indicate that the hybrid algorithm provided the best results *ex aequo* with another algorithm(s).

As can be seen, the hybrid algorithm provides better (or at least equal) results in all cases except for the instance $0.75 \times 30 \times 250$, for which the MA obtains the best result and the best mean corresponds to the hybrid algorithm. For the smallest problem instances, results for the MA and hybrid algorithm coincide. This may be due to the lower difficulty of these instances; the search overhead of switching from the MA to BS may not be worth it in this case. The hybrid algorithm only becomes advantageous in larger instances, where the MA faces a more difficult optimization scenario. The statistical significance of the results has been evaluated using a non-parametric test – the Wilcoxon ranksum test. It has been found that for $n = 100$ results for the MA and hybrid algorithm coincide, for $n = 250$ statistical significance is found for $\alpha = 0.25, m = 30$, and for $n = 500$ differences are statistically significant for all cases except $m = 30, \alpha = 0.75$.

With respect to the BS algorithm, notice that the hybrid algorithm always provides better solutions than (or at least equal to) beam search. Figure 1a shows the evolution of the best value found by the different algorithms for a specific problem instance ($\alpha = 0.25$, $m = 30$, $n = 500$). As can be seen, the hybrid algorithm outperforms both the MA and BS due to their synergetic combination.

The best known results for these instances are currently due to two hybrid ILP\tabu search algorithms presented in [22] and [23]. For the $0.25 \times 30 \times 500$ and $0.25 \times 10 \times 500$ instances, our hybrid algorithm obtains better results than the ones presented in [22], whereas for the $0.75 \times 30 \times 500$ and $0.25 \times 10 \times 500$ instances results obtained by [22] are 1 and 6 units better. Results in [23] could not be reached. Note however that to obtain those results, the total running time for an instance with $m = 30$ was 12 hours for [22] and 33 hours for [23], whilst our hybrid algorithm was only given 4.16 hours (accumulating the 25 runs).

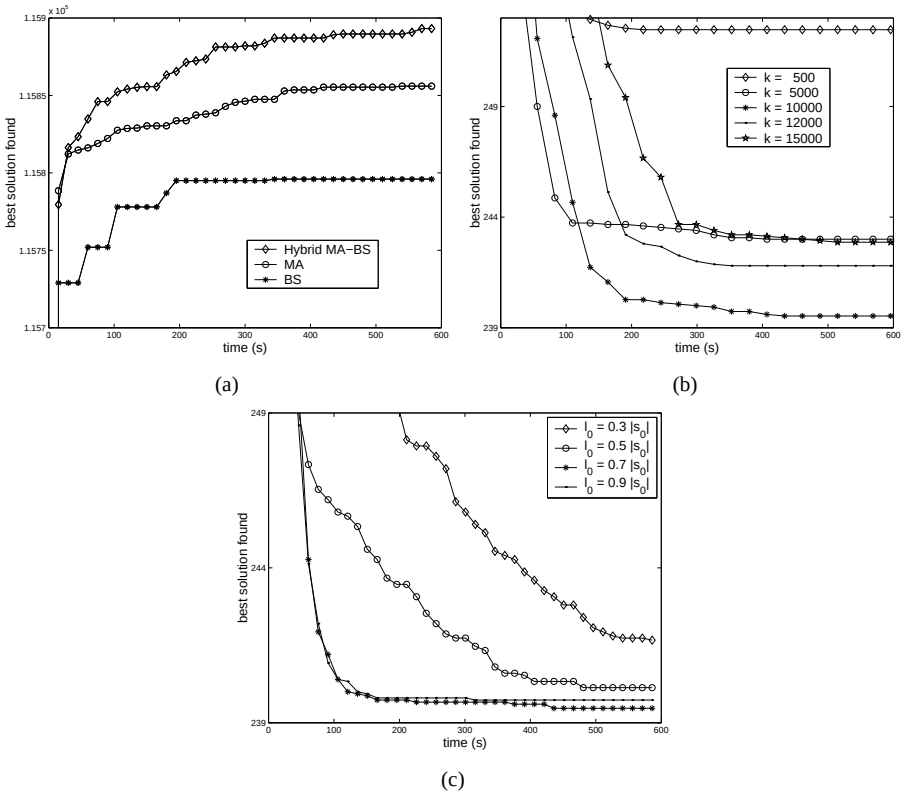

Fig. 1: (a) Evolution of the best solution in the MA, beam search, and the hybrid algorithm during 600 seconds of execution for a MKP problem instance with $\alpha = 0.25$, $m = 30$, $n = 500$. Curves are averaged for 25 runs in the case of the MA and the hybrid algorithm. (b) Evolution of the best solution in the hybrid algorithm during 600 seconds of execution for a random SCSP problem instance average over different values for $|\Sigma|$ for different values for k. (c) Evolution of the best solution in the hybrid algorithm during 600 seconds of execution for a random SCSP problem instance average over different values for $|\Sigma|$ for different values for l_0.

B. Experimental Results for the SCSP

A hybrid algorithm for the SCSP was obtained from the general description in Section II with the following parameters: $k = 10000$, $l = 10$ and $popsize = 100$ (again see Section IV-C for details).

Selection on every level of the search tree used the following quality function to measure a tentative solutions s^t in each node:

$$quality\ (s^t, L) \ \triangleq \ \sum_{s_i \in L} \{\, |s_i^e| \mid (s_i^e, r_i) = s^t \gg s_i \} \tag{7}$$

In this way, tentative solutions embedding more symbols in L are selected. Let us note that all tentative solutions in one level of the search tree have the same length, so the algorithm selects nodes that provide good initial segments for constructing a short supersequence. Before being injected into the MA population, tentative solutions were randomly completed and repaired using the function described in Sect. III-B.3.

TABLE II: Results of the different heuristics for the SCSP.

	MM			WMM		
$\|\Sigma\|$	best	mean $\pm$ σ	i.%	best	mean $\pm$ σ	i.%
2	112.0	112.0 $\pm$ 0.1	0.0	114.8	114.8 $\pm$ 0.0	-2.5
4	152.6	153.4 $\pm$ 0.7	0.0	157.8	157.8 $\pm$ 0.0	-2.8
8	212.4	213.8 $\pm$ 0.9	0.0	208.2	208.2 $\pm$ 0.0	2.6
16	283.8	286.1 $\pm$ 2.0	0.0	272.8	273.4 $\pm$ 0.5	4.4
24	330.2	333.9 $\pm$ 2.3	0.0	324.0	325.2 $\pm$ 0.7	2.6

	AL			MA		
$\|\Sigma\|$	best	mean $\pm$ σ	i.%	best	mean $\pm$ σ	i.%
2	121.4	123.4 $\pm$ 2.0	-10.2	111.2	112.4 $\pm$ 1.0	-0.4
4	183.0	191.2 $\pm$ 4.7	-24.6	150.0	152.6 $\pm$ 1.8	0.5
8	252.2	276.8 $\pm$ 6.4	-29.5	203.0	205.5 $\pm$ 1.9	3.9
16	320.6	352.9 $\pm$ 7.4	-23.3	264.4	270.4 $\pm$ 3.7	5.5
24	363.8	390.8 $\pm$ 6.4	-17.0	301.8	305.6 $\pm$ 3.4	8.5

	Hybrid MA-BS			BS		
$\|\Sigma\|$	best	mean $\pm$ σ	i.%	best	mean $\pm$ σ	i.%
2	110.6	110.7 $\pm$ 0.0	1.2	110.8	110.8 $\pm$ 2.2	1.1
4	145.6	146.4 $\pm$ 0.5	4.6	149.4	149.4 $\pm$ 3.6	2.6
8	191.6	192.6 $\pm$ 1.4	9.9	198.2	198.2 $\pm$ 1.9	7.3
16	242.8	244.0 $\pm$ 1.0	14.7	253.0	253.0 $\pm$ 5.3	11.6
24	280.2	281.2 $\pm$ 0.8	15.8	287.8	287.8 $\pm$ 1.7	13.8

For the MA and the hybrid algorithm, local search was randomly applied with a probability of $p = 0.01$. Preliminary experiments were conducted with values $p \in \{0, 0.01, 0.1, 0.5, 1\}$, and this particular setting provided a better tradeoff between the attainable improvement, and the additional computational cost implied [24].

Two different sets of problem instances have been used in the experimentation. The first one is composed of random strings with different lengths. To be precise, each instance is composed of eight strings, four of them with 40 symbols, and the remaining four with 80 symbols. Each of these strings is randomly built, using an alphabet Σ. Five subsets of instances have been defined using different alphabet sizes, namely $|\Sigma|$ =2, 4, 8, 16 and 24. For each alphabet size, five different instances have been generated.

The second set of instances constitutes a more realistic benchmark, and consists of strings with a common source. A DNA sequence from a SARS coronavirus strain has been retrieved from a genomic database[1], and taken as supersequence; then, different sequences were obtained from this supersequence by scanning it from left to right, skipping nucleotides with a certain fixed probability. In these experiments, the supersequence is 158-nucleotide long, the gap probability is 10%, 15%, or 20% and the number of so-generated sequences is 10.

First of all, the results for random strings are shown in Table II. The results of AL are averaged over all permutations of the alphabet (or a maximum 100,000 permutations for $|\Sigma| \geqslant 16$). For MM and WMM, a restart approach allowing 600 seconds per run was used. The results of WW, WMM, MA and the hybrid algorithm are

TABLE III: Results of the different heuristics on the SARS DNA sequence.

	MM		WMM	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	158	158.0 $\pm$ 0.0	158	158.0 $\pm$ 0.0
15%	160	160.0 $\pm$ 0.0	231	231.0 $\pm$ 0.0
20%	228	229.6 $\pm$ 1.8	266	266.0 $\pm$ 0.0

	AL		MA	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	307	315.2 $\pm$ 6.8	158	158.0 $\pm$ 0.0
15%	293	304.3 $\pm$ 8.8	158	159.0 $\pm$ 2.8
20%	274	288.3 $\pm$ 8.6	159	177.0 $\pm$ 9.3

	Hybrid MA-BS		BS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	158	158.0 $\pm$ 0.0	158	n.a.
15%	158	158.0 $\pm$ 0.0	158	n.a.
20%	158	158.0 $\pm$ 0.0	158	n.a.

averaged over 5 runs and the results of BS are for a single run. In all cases, the results are further averaged over 5 different problem instances (thus, 25 runs per each alphabet size). The column labelled as "i.%" indicates the average relative improvement of the mean solutions with respect to the mean solution provided by MM. Clearly, the hybrid algorithm performs better than the other algorithms on all the tests followed by BS and MA. This superior performance is statistically significant in all cases, and indicates the synergy of this combination; thus, it supports the idea that it is a profitable approach for tackling this combinatorial problem. WMM performs better than MM for larger alphabet sizes whereas AL obtains the worst results.

Finally, the results for the strings from the SARS DNA sequence are shown in Table III. Again, AL performs quite poorly here whilst MM and WMM match the optimal solution for low gap probability (10%). Note that WMM performs worse than MM for this problem. BS and the hybrid algorithm find the optimal solution for three instances whereas MA misses the last one.

To the best of our knowledge, the GA described in [19] and an Ant Colony Optimisation algorithm (ACO) in [25] are among the best heuristics for the SCSP. A comparison of our algorithm with [19], indicates that the hybrid algorithm performs similarly for $|\Sigma| \leqslant 4$, and it is 5 points better (taking the performance of MM as baseline) for $|\Sigma| = 16$. No tests were performed there for $|\Sigma| \in \{8, 24\}$, hence we can not compare our algorithm to theirs for these cases. With respect to [25], their best algorithm performs 3.4 points better than our hybrid algorithm for a random problem instance with $|\Sigma| = 16$, but it needs about 2 hours on a Pentium II 300MHz, while ours needs only about 200 seconds on a machine 8 times faster.

C. Dynamics and Sensitivity analysis on the Hybrid Model

After analyzing the traces of the algorithms, we have observed that most of the improved solutions obtained by the hybrid algorithm are contributed by the MA part (e.g., 37% of the invocations of the MA result in a strictly improved incumbent solution for the SCSP; this figure is just 10% for BS). On the other hand, the role of the BS part seems to periodically provide good candidate solutions to the MA population. This is confirmed by the better results obtained by the hybrid algorithm and the fact that the total number of generations in the MA part needed to obtain these results is considerably less (for example, the MA needs an average of 30815 generations to obtain its best result for a random SCSP instance with $|\Sigma| = 24$, whilst the MA part in the hybrid algorithm only needs 18055 generations on average). Recently, we have applied this hybrid approach to other combinatorial optimization problems (finding minimum weight ultrametric trees [26] and maximum density still lifes [27]), and have obtained similar results.

Several parameters can be tuned to control the hybrid algorithm, namely parameters l_0 and l (controlling the balance between MA and BS) and parameter k (which can be used to adjust the search breadth in BS). Aiming to determine good values for these parameters, a sensitivity analysis was done for both the MKP and the SCSP. We detected that the value of parameter k is directly related to the effort of computing bounds. Increasing this parameter undoubtedly improves the quality of the results obtained by the BS part of the algorithm, but may delay the descent through the search tree, specially when the estimation of bounds for each node is computationally expensive. If this is the case (e.g., for the MKP) a low value for k should be used. Certainly, this value might be increased, but a longer execution time would be needed for the algorithm to make progress. Regarding the SCSP, computation of bounds is simple, so a greater value for k parameter may be used. Figure 1b shows the average evolution of the best solution to this problem for the hybrid algorithm for different values of k. Clearly, the quality of the solution is increased with higher values for k up to 10000. For greater values of k, the performance of the hybrid algorithm degrades as most of the computation is dedicated to the BS part of the algorithm.

Parameters l_0 and l are related to bound tightness, so that when bounds are tight and the BnB makes fast progress, it is suggested that low values be used for these parameters. This is the case in the MKP. In the SCSP, bounds are not tight and we observed that good solutions were only obtained after descending a substantial number of levels in the BS tree (Figure 1c shows the average evolution of the hybrid algorithm for different values for l_0). For this reason an estimation for the SCSP solution s_0 was calculated using the WMM algorithm and its length was used to set $l_0 = 0.7 \cdot |s_0|$ in our experiments.

V. Conclusions and Future Work

We have presented a model for the hybridization of MAs with a truncated BnB algorithm. The model tries to boost performance by mutual collaboration: the MA provides improved bounds that the BnB algorithm can use to purge the problem queue, whereas the BnB guides the MA to look into promising regions of the search space.

The resulting hybrid algorithm has been tested on two combinatorial optimization problems –the MKP and the SCSP– with encouraging results: the hybrid algorithm produces better results than the constituent algorithms on

specific instances. This indicates the synergy of this combination, thus supporting the idea that it is a profitable approach for tackling difficult combinatorial problems. In this sense, future work will be directed to confirm these findings on additional combinatorial problems, as well as developing extensions of the model retaining completeness.

Another very interesting line for future developments relates to the parallelization of the hybrid algorithm. Several possibilities can be considered here. On the one hand, any of the constituent algorithms could be internally parallelized; for instance, the MA could use an island-based model, or we could distribute the evaluation of nodes in the current level of the BS among a number of computers. This may well lead to improved results due to the numerical speed-up in the BnB part, and the algorithmic speed-up in the MA component [28]. On the other hand, a fully asynchronous functioning of both components (MA and BnB) with occasional communication is a very appealing option for dealing with large-scale optimization tasks. The analysis of this asynchronous model, as well as the design of alternative collaboration architectures promises to be a fruitful line for future research.

REFERENCES

[1] E. Lawler and D. Wood, "Branch and bounds methods: A survey," *Operations Research*, vol. 4, no. 4, pp. 669–719, 1966.

[2] J. T. Linderoth and W. P. Savelsbergh, "A computational study of search strategies in mixed integer programming," *INFORMS Journal on Computing*, vol. 11, pp. 173–187, 1999.

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

[4] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] P. Moscato and C. Cotta, "A gentle introduction to memetic algorithms," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Boston MA: Kluwer Academic Publishers, 2003, pp. 105–144.

[6] W. Hart, N. Krasnogor, and J. Smith, *Recent Advances in Memetic Algorithms*, ser. Studies in Fuzziness and Soft Computing. Berlin Heidelberg: Springer-Verlag, 2005, vol. 166.

[7] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," California Institute of Technology, Pasadena, California, USA, Tech. Rep. Caltech Concurrent Computation Program, Report. 826, 1989.

[8] J. Puchinger and G. Raidl, "Combining metaheuristics and exact algorithms in combinatorial optimization: a survey and classification," in *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, ser. Lecture Notes in Computer Science, J. Mira and J. Álvarez, Eds., vol. 3562. Berlin Heidelberg: Springer-Verlag, 2005, pp. 41–53.

[9] A. Barr and E. Feigenbaum, *Handbook of Artificial Intelligence*. New York NY: Morhan Kaufmann, 1981.

[10] B. Korte and R. Schrader, "On the existence of fast approximation schemes," in *Nonlinear Programming 4*, O. Mangasarian, R. Meyer, and S. Robinson, Eds. Academic Press, 1981, pp. 415–437.

[11] E. Balas and H. Martin, "Pivot and Complement - a heuristic for 0-1 programming," *Management Science*, vol. 26, no. 1, pp. 86–96, 1980.

[12] P. Chu and J. Beasley, "A genetic algorithm for the multidimensional knapsack problem," *Journal of Heuristics*, vol. 4, pp. 63–86, 1998.

[13] C. Cotta and J. Troya, "A hybrid genetic algorithm for the 0-1 multiple knapsack problem," in *Artificial Neural Nets and Genetic Algorithms*, G. Smith, N. Steele, and R. Albrecht, Eds. Wien New York: Springer-Verlag, 1998, pp. 251–255.

[14] G. Raidl and J. Gottlieb, "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem." *Evolutionary Computation*, vol. 13, no. 4, pp. 441–475, 2004.

[15] H. Ishibuchi, S. Kaige, and K. Narukawa, "Comparison between Lamarckian and Baldwinian repair on multiobjective 0/1 knapsack problems," in *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005*, ser. Lecture Notes in Computer Science, C. Coello Coello, A. Hernández Aguirre, and E. Zitzler, Eds., vol. 3410, Berlin Heidelberg, 2005, pp. 370–385.

[16] M. Middendorf, "More on the complexity of common superstring and supersequence problems," *Theoretical Computer Science*, vol. 125, pp. 205–228, 1994.

[17] K. Pietrzak, "On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems," *Journal of Computer and System Sciences*, vol. 67, no. 1, pp. 757–771, 2003.

[18] C. Fraser, “Subsequences and supersequences,” Ph.D. dissertation, University of Glasgow, Department of Computer Science, 1995.

[19] J. Branke, M. Middendorf, and F. Schneider, “Improved heuristics and a genetic algorithm for finding short supersequences,” *OR-Spektrum*, vol. 20, pp. 39–45, 1998.

[20] S. Rahmann, “The shortest common supersequence problem in a microarray production setting,” *Bioinformatics*, vol. 19, no. Suppl. 2, pp. ii156–ii161, 2003.

[21] C. Cotta, “A comparison of evolutionary approaches to the shortest common supersequence problem,” in *Computational Intelligence and Bioinspired Systems*, ser. Lecture Notes in Computer Science, J. Cabestany, A. Prieto, and D. Sandoval, Eds., vol. 3512. Berlin Heidelberg: Springer-Verlag, 2005, pp. 50–58.

[22] M. Vasquez and J. Hao, “A hybrid approach for the 0–1 multidimensional knapsack problem,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001, pp. 328–333.

[23] M. Vasquez and Y. Vimont, “Improved results on the 0–1 multidimensional knapsack problem,” *European Journal of Operational Research*, vol. 165, pp. 70–81, 2005.

[24] C. Cotta, “Memetic algorithms with partial lamarckism for the shortest common supersequence problem,” in *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, ser. Lecture Notes in Computer Science, J. Mira and J. Álvarez, Eds., vol. 3562. Berlin Heidelberg: Springer-Verlag, 2005, pp. 84–91.

[25] M. Michel, R.; Middendorf, “An ant system for the shortest common supersequence problem,” in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London: McGraw-Hill, 1999, pp. 51–61.

[26] B. Wu, K.-M. Chao, and C. Tang, “Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices,” *Journal of Combinatorial Optimization*, vol. 3, no. 2, pp. 199–211, 1999.

[27] R. Bosch, “Stable patterns in variants of Conway’s game of life,” *Operations Research Letters*, vol. 27, no. 1, pp. 7–11, 2000.

[28] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (now Springer-Verlag), 2000, vol. 1.

LIST OF FIGURES

LIST OF TABLES

LIST OF FOOTNOTES