A Comparison of Evolutionary Approaches to the Shortest Common Supersequence Problem

Carlos Cotta

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática,
University of Málaga, Campus de Teatinos, 29071 - Málaga, SPAIN

ccottap@lcc.uma.es

Abstract. The Shortest Common Supersequence problem is a hard combinatorial optimization problem with numerous practical applications. Several evolutionary approaches are proposed for this problem, considering the utilization of penalty functions, GRASP-based decoders, or repairing mechanisms. An empirical comparison is conducted, using an extensive benchmark comprising problem instances of different size and structure. The empirical results indicate that there is no single best approach, and that the size of the alphabet, and the structure of strings are crucial factors for determining performance. Nevertheless, the repair-based EA seems to provide the best performance tradeoff.

1 Introduction

The Shortest Common Supersequence (SCS) problem is a classical problem from the realm of string analysis. Roughly speaking, the SCS problem amounts to finding a minimal-length sequence S of symbols such that every string in a certain set L can be generated from S by removing some symbols of the latter. The resulting combinatorial problem is enormously interesting, not only from the point of view of Theoretical Computer Science, but also from an applied perspective. Indeed, it has applications in planning, data compression, and bioinformatics among other fields [1–3].

Unfortunately, the SCS problem has been shown to be hard under various formulation and restrictions [4–6] (a summary of these hardness results is provided in Sect. 2). This way, although exact approaches have been proposed to tackle this problem (see e.g., [7]), these are impractical for even moderate-size problem instances. Hence, heuristic approaches are in order. In this sense, greedy approaches have been popular. For example, one can cite the Majority Merge (MM) algorithm, and related variants [8]. However, these heuristics are not the ultimate solvers for this problem due to their myopic functioning. More sophisticated techniques such as evolutionary algorithms (EAs) can be used to overcome the limitations of greedy techniques.

This work will analyze and compare four different evolutionary approaches to the SCS problem, involving either a direct search in the space of supersequences, or using auxiliary search spaces and more complex decoding mechanisms for obtaining high-quality solutions.

2 The Shortest Common Supersequence Problem

Let us start by formally defining the SCS problem in its decisional version:

Shortest Common Supersequence Problem
Instance: A set L of m strings $\{s_1, \cdots, s_m\}$ or arbitrary length over an alphabet Σ (i.e., $s_i \in \Sigma^*$, for $1 \leqslant i \leqslant m$), and a positive integer k.
Question: Does there exist a string $s \in \Sigma^*$, $|s| \leqslant k$, such that s is a supersequence[1] of every $s_i \in L$?

Having defined the problem, let us now consider its computational complexity.

2.1 Complexity Results for the SCS Problem

The SCS problem can be shown to be NP-hard, even if strong constraints are posed on L, or on Σ. For example, it is NP-hard in general when all s_i have length two [2], or when the alphabet size $|\Sigma|$ is two [5]. It must be noted that –despite being important– NP-hard results are usually over-stressed; in fact, there are many problems that can be efficiently solved in practice, yet they are NP-hard.

Parameterized complexity [9] tries to deal with this issue, providing a more sensible characterization of hardness. The key idea is to isolate hardness (i.e., non-polynomial behavior) within a certain set of parameters. This way, if these parameters are kept fixed, the problem can be efficiently[2] solved for large problem sizes. Vertex Cover is a good example of this situation: it is NP-hard, but it can be solved in linear time in the number of vertices, when the size of the vertex cover sought is kept fixed. Problems such as Vertex Cover for which this hardness-isolation is possible are termed *fixed-parameter tractable* (FPT). Non-FPT problem will fall under some class in the $W-$hierarchy. Hardness for the parameterized class $W[1]$ is the current measure of intractability.

Several parameterizations are possible for the SCS problem. Firstly, the maximum length k of the supersequence sought can be taken as a parameter. If the alphabet size is constant, or another parameter, then the problem turns in this case to be FPT, since there are at most $|\Sigma|^k$ supersequences. However, this is not very useful in practice because $k \geqslant \max |s_i|$. If the number of strings m is used as parameter, then SCS is $W[1]-$hard, and remains so even if $|\Sigma|$ is taken as another parameter [3], or is constant [6]. Failure of finding FPT results in this latter scenario is particularly relevant since the alphabet size in biological problems is fixed (e.g., there are just four nucleotides in DNA). Furthermore, notice that absence of FPT algorithms implies the non-existence of fully polynomial time approximation schemes (FPTAS) for the corresponding problem.

[1] s is a supersequence of r if $r = \epsilon$ (where ϵ is the empty string) or if $s = \alpha s'$, $r = \alpha' r'$ and s' is a supersequence of r' ($\alpha = \alpha'$) or s' is a supersequence of r ($\alpha \neq \alpha'$).

[2] In time $O(f(k)n^c)$, where k is the parameter, n is the problem size, f is an arbitrary function of k only, and c is a constant independent of k and n.

2.2 Heuristics for the SCS Problem

The hardness results mentioned in the previous subsection 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 strings in L, and removing these symbols from the corresponding strings. More precisely:

```
Heuristic MM ($L = \{s_1 \cdots, s_m\}$)
1. let $s \leftarrow \epsilon$
2. do
   (a) for $\alpha \in \Sigma$ do let $\nu(\alpha) \leftarrow \sum_{s_i = \alpha s'_i} 1$
   (b) let $\beta \leftarrow \max^{-1}\{\nu(\alpha) \mid \alpha \in \Sigma\}$
   (c) for $s_i \in L, s_i = \beta s'_i$ do let $s_i \leftarrow s'_i$
   (d) let $s \leftarrow s\beta$
   until $\sum_{s_i \in L} |s_i| = 0$
```

The myopic functioning of MM makes it incapable of grasping the global structure of strings in L though. In particular, MM misses the fact that the strings can have different lengths [8]. This implies that symbols at the front of short strings will have more chances to be removed, since the algorithm has still to scan the longer strings. For this reason, it is less urgent to remove those symbols. In other words, it is better to concentrate in shortening longer strings first. This can be done by assigning a weight to each symbol, depending of the length of the string in whose front is located. Branke *et al.* [8] propose to use precisely this string length as weight, i.e., step 2a in the previous pseudocode would be modified to have $\nu(\alpha) \leftarrow \sum_{s_i = \alpha s'_i} |s'_i|$. This modified heuristic will be termed weighted MM (WMM).

Several other heuristics were also defined in [8] on the basis of WMM. For example, one of them has $\nu(\alpha) \leftarrow |\text{WMM}(\{s_1|_\alpha, \cdots, s_m|_\alpha\})|$ (to be minimized), where $s|_\alpha$ is the string obtained by removing α from the front of s; ties are broken by maximizing $|\text{WMM}(\{s_i \mid s_i = \alpha s'_i\})|$. A more interesting heuristic results from the combination of EAs and WMM. In this heuristic, the EA is used to evolve weights for each character of every string. These weights are utilized within step 2a, modifying the influence of symbols in each string. This is done by multiplying the WMM weight with the evolved weight, i.e., $\nu(\alpha) \leftarrow \sum_{s_i = \alpha s'_i} w_{p_{\alpha i}, i} |s'_i|$, where $p_{\alpha i}$ is the position of the current front symbol α in the original string s_i. In a further refinement, the EA is used also to evolve a basic value to be added to each weight before evaluation.

This EA approach is similar to the EA used in [10] for the multidimensional knapsack problem, in which a greedy heuristic was used to generate solutions, and weights were evolved in order to modify the value of objects (thus making the underlying heuristic take different decisions). Next section will explore alternative EA definitions in which the search is conducted in search spaces different to this weight space.

3 Evolutionary Approaches to the SCS Problem

Clearly, one of the difficulties faced by an EA when applied to the SCS problem 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. 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. These three possibilities will be explored below.

Let us firstly consider the simplest version, namely the use of a penalty function. The idea is to have an EA evolving strings in Σ^*, using the supersequence length as the quality measure for feasible solutions, and adding an extra penalty term for infeasible solutions. This has been implemented as follows: let s be the tentative solution provided by the EA, and let $L = \{s_1, \cdots s_m\}$ be the target strings; then, the fitness (to be minimized) is

$$fitness\,(s, L) = \begin{cases} 0 & \text{if } \forall i : s_i = \epsilon \\ 1 + fitness(s', L|_\alpha) & \text{if } \exists i : s_i \neq \epsilon \text{ and } s = \alpha s' \\ |\mathrm{MM}(L)| & \text{if } \exists i : s_i \neq \epsilon \text{ and } s = \epsilon \end{cases} \tag{1}$$

This algorithm will be termed Penalty EA, and relies in MM for providing a heuristic assessment on how much longer a string should be to account for uncovered string suffixes in L. An obvious variant consists of injecting the actual sequence returned by MM back to the candidate solution. Thus, the sequence is repaired, resulting in a feasible solution. More precisely, the repairing function $\rho : \Sigma^* \times (\Sigma^*)^m \to \Sigma^*$ can be described as follows:

$$\rho\,(s, L) = \begin{cases} s & \text{if } \forall i : s_i = \epsilon \\ \rho(s', L) & \text{if } \exists i : s_i \neq \epsilon \text{ and } \nexists i : s_i = \alpha s'_i \text{ and } s = \alpha s' \\ \alpha\rho(s', L|_\alpha) & \text{if } \exists i : s_i = \alpha s'_i \text{ and } s = \alpha s' \\ \mathrm{MM}(L) & \text{if } \exists i : s_i \neq \epsilon \text{ and } s = \epsilon \end{cases} \tag{2}$$

As it can be seen, this repairing function not only completes s in order to have a valid supersequence, but also removes intermediate symbols that are not present at the front of any string at a certain step. Thus, it also serves the purpose of local improver. This algorithm is termed Repair EA.

The third approach is to have the EA handling uniquely feasible solutions. In this case, we have considered the utilization of an auxiliary space, and a *smart* decoder in order to perform the mapping to the sequence space. This decoder can be based on either MM or WMM, and borrows some ideas from greedy randomized adaptive search procedures (GRASP). This latter metaheuristic also relies on an underlying greedy heuristic, and proceeds iteratively by selecting at each step an attribute of the solution from a *candidate list* [11]. This selection can be typically done by using a qualitative criterion (i.e., a candidate is selected among the best k elements in the candidate list, k being a parameter), or a quantitative criterion (i.e., a candidate is selected among the elements whose quality is within a certain range).

One of the potential problems of the basic GRASP procedure described before relies on the selection of the parameter for selecting an attribute value from the candidate list. As shown in [12], using a single fixed value for this parameter may hinder finding high-quality solutions. Several options are possible to solve this problem. On one hand, a learning-based strategy termed *reactive* GRASP was proposed [13]. On the other hand, the utilization of EAs to evolve the sequence of selection decisions was presented in [14]. This latter approach is precisely considered here. To be precise, the EA is used to evolve a sequence of integers $\langle\delta_1, \cdots, \delta_n\rangle$, $\delta_i \in [1..|\Sigma|]$, where $n = \sum_{i=1}^{m} |s_i|$. At each step of the decoding process, a ranked list of the potential symbols to be added to the supersequence is constructed using either the MM or the WMM criterion; the value δ_i indicates that the δ_i-th best symbol is chosen at the i-th step. Notice that the construction of the supersequence will in general be accomplished in many less steps than n.

4 Experimental Validation

The experiments have been done with a steady-state EA ($popsize = 100$, $p_X = .9$, $p_m = 1/n$, $maxevals = 100,000$), using binary tournament selection, uniform crossover, and random-substitution mutation. Three different sets of problem instances have been considered 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. Four subsets of instances have been defined using different alphabet sizes, namely $|\Sigma|$ =2, 4, 8, and 16. For each alphabet size, five different instances have been generated.

The second set of instances comprises strings whose structure is deceptive for greedy heuristics such as MM or WMM. These are the following (cf. [8]):

- $L_1 = \{9 \times a^{40}, 4 \times ba^{39}, 2 \times bba^{38}, 1 \times bbba^{37}\}$. The optimal solution for this instance is to firstly remove the three b's, and then remove the forty a's. It thus has length 43.
- $L_2 = \{9 \times a^{40}, 4 \times b^{13}a^{27}, 2 \times b^{26}a^{14}, 1 \times b^{39}a\}$. This instance is similar to L_1, and again the optimal solution is to firstly remove the b's, i.e., $b^{39}a^{40}$, a solution of length 79.
- $L_3 = \{8 \times a^{20}b^{20}, 8 \times b^{20}c^{20}\}$. The optimal solution in this case is $a^{20}b^{20}c^{20}$.

Finally, a more realistic benchmark consisting of strings with a common origin has been considered. A DNA sequence from a SARS coronavirus strain has been retrieved from a genomic database[3], and has been taken as supersequence; then, different sequences are obtained from this supersequence by scanning it from left to right, and skipping nucleotides with a certain fixed probability. In these experiments, the length of the supersequence is 158, the gap probability is 10%, 15%, or 20% and the number of so-generated sequences is 10.

[3] `http://gel.ym.edu.tw/sars/genomes.html`, accession AY271716.

Table 1. Results of the different heuristics on 8 random strings (4 of length 40, and 4 of length 80), for different alphabet sizes $|\Sigma|$. The results of MM and WMM are averaged over 150 executions, and the results of the EAs are averaged over 30 runs. In all cases, the results are further averaged over five different problem instances.

	MM		WMM		Penalty EA	
$\|\Sigma\|$	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
2	115.0	120.46 ± 2.16	114.8	116.13 ± 0.89	119.0	123.61 ± 2.52
4	164.2	175.47 ± 5.02	157.8	161.85 ± 2.81	200.2	223.31 ± 12.18
8	227.0	249.33 ± 6.92	210.4	219.61 ± 4.90	366.8	445.94 ± 23.48
16	309.2	333.57 ± 9.08	282.8	296.07 ± 5.31	538.2	569.74 ± 11.73
	Repair EA		GRASP-EA(MM)		GRASP-EA(WMM)	
$\|\Sigma\|$	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
2	111.2	112.58 ± 0.75	113.0	116.95 ± 1.97	110.8	113.31 ± 1.49
4	151.6	155.17 ± 1.85	160.6	167.47 ± 3.11	151.6	157.81 ± 2.98
8	205.4	213.47 ± 3.97	217.6	228.18 ± 4.22	204.0	211.01 ± 3.05
16	267.0	281.81 ± 5.88	286.2	297.53 ± 4.94	271.2	278.15 ± 2.97

Table 2. Results of the different heuristics on the deceptive problem instances. The results of MM and WMM are averaged over 150 executions, and the results of the EAs are averaged over 30 runs.

	MM		WMM		Penalty EA	
	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
L_1	157	157.00 ± 0.00	92	92.00 ± 0.00	43	43.00 ± 0.00
L_2	121	121.00 ± 0.00	91	91.00 ± 0.00	79	79.00 ± 0.00
L_3	68	77.73 ± 2.98	79	79.46 ± 0.50	60	60.07 ± 6.42
	Repair EA		GRASP-EA(MM)		GRASP-EA(WMM)	
	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
L_1	43	43.00 ± 0.00	43	43.00 ± 0.00	43	43.00 ± 0.00
L_2	79	79.00 ± 0.00	79	79.00 ± 0.00	79	79.00 ± 0.00
L_3	60	60.00 ± 0.00	60	60.00 ± 0.00	60	60.00 ± 0.00

Table 3. Results of the different heuristics on the strings from the SARS DNA sequence for different gap probabilities. The results of MM and WMM are averaged over 150 executions, and the results of the EAs are averaged over 30 runs.

	MM		WMM		Penalty EA	
gap%	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
10%	158	158.00 ± 0.00	158	158.00 ± 0.00	350	428.00 ± 26.83
15%	158	158.00 ± 0.00	158	158.00 ± 0.00	358	397.87 ± 20.86
20%	160	227.77 ± 25.75	231	234.41 ± 1.88	339	377.30 ± 18.87
	Repair EA		GRASP-EA(MM)		GRASP-EA(WMM)	
gap%	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
10%	158	158.00 ± 0.00	163	181.60 ± 14.55	163	171.97 ± 8.42
15%	158	158.00 ± 0.00	199	218.70 ± 8.83	191	212.60 ± 9.04
20%	165	180.80 ± 15.73	205	222.47 ± 8.88	212	222.27 ± 5.37

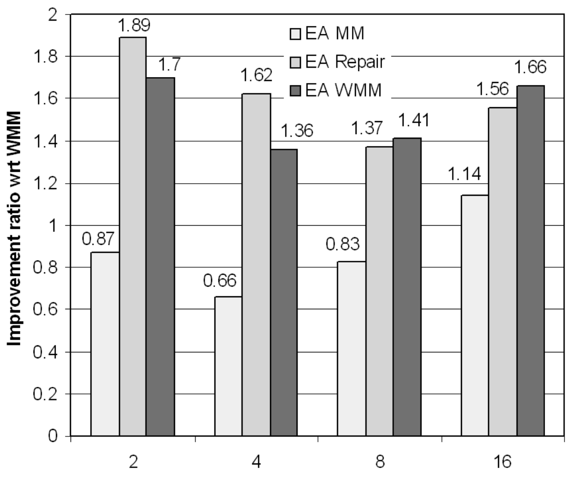

Fig. 1. Improvement ratios of the different EAs with respect to WWM for different alphabet sizes.

First of all, the results for random strings are shown in Table 1. WMM performs better than MM, and both the repair-based EA and the WMM-based EA outperform the remaining algorithms. The MM-based EA is capable of beating MM, and produce similar results to those of WMM; however, it cannot compete with the repair-based EA or the WMM-based EA. Notice also the poor results of the plain Direct EA (it cannot even produce a feasible solution for $|\Sigma| = 16$).

It is interesting to note the U-shaped behavioral pattern of the EAs with respect to the basic heuristics. This is illustrated in Fig. 1, that shows the ratio between the percentage of improvement of the EAs (with respect to MM), and the percentage of improvement of WMM (also with respect to MM). The pattern for the MM-based EA and the WMM-based EA is the same (save for the scale factor) with a minimum at $|\Sigma| = 4$; however, this pattern is shifted for the repair-based EA that has its minimum at $|\Sigma| = 8$. At any rate, the performance of the repair-based EA is here very similar to that of the WMM-based EA.

The results for the deceptive problem instances are shown in Table 2. As expected, both MM and WMM are fooled by the problem structure. However, the EAs are capable of finding consistently the optimal solutions without major difficulties. This emphasizes the important role played by the evolutionary search in this problem.

Finally, the results for the strings from the SARS DNA sequence are shown in Table 3. The basic heuristics perform quite well for low gap probability (i.e., for larger strings). However, neither the MM-based EA nor the WMM-based EA can match this performance (recall that $|\Sigma| = 4$, that is, the lower performance point for these EAs). For the larger gap probability there is a remarkable performance drop of the basic heuristics, and GRASP-based EAs can catch up with these. The repair-based EA offers the best results throughout the three problem instances, performing significantly better than the remaining algorithms.

5 Conclusions

Four different EAs have been proposed and compared for the SCS problem. The main goal has been to determine which of the typical constraint-handling procedures is more appropriate for this problem. The experimental results seem to indicate that the best performance tradeoff is provided by the repair-based EA: it behaves much better than the other EAs in some problem instances, and similarly to these in the remaining ones. The GRASP-based EAs are located at the next performance level, the WMM-based EA performing better than the MM-based EA. Future work will be directed to test other underlying heuristics for decoding solutions. The repair-based EA seems to be the adversary with which such new EAs should be confronted.

Acknowledgements. This work is partially supported by Spanish MCyT and FEDER under contract TIC2002-04498-C05-02.

References

1. Foulser, D., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence **57** (1992) 143–181
2. Timkovsky, V.: Complexity of common subsequence and supersequence problems and related problems. Cybernetics **25** (1990) 565–580
3. Hallet, M.: An integrated complexity analysis of problems from computational biology. PhD thesis, University of Victoria (1996)
4. Bodlaender, H., Downey, R., Fellows, M., Wareham, H.: The parameterized complexity of sequence alignment and consensus. Theoretical Computer Science **147** (1994) 31–54
5. Middendorf, M.: More on the complexity of common superstring and supersequence problems. Theoretical Computer Science **125** (1994) 205–228
6. Pietrzak, K.: On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. Journal of Computer and System Sciences **67** (2003) 757–771
7. Fraser, C.: Subsequences and Supersequences. PhD thesis, University of Glasgow, Department of Computer Science (1995)
8. Branke, J., Middendorf, M., Schneider, F.: Improved heuristics and a genetic algorithm for finding short supersequences. OR-Spektrum **20** (1998) 39–45
9. Downey, R., Fellows, M.: Parameterized Complexity. Springer-Verlag (1998)
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, Wien New York, Springer-Verlag (1998) 251–255
11. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization **6** (1995) 109–133
12. Prais, M., Ribeiro, C.: Parameter variation in GRASP procedures. Investigación Operativa **9** (2000) 1–20
13. Prais, M., Ribeiro, C.: Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing **12** (2000) 164–176
14. Cotta, C., Fernández, A.: A hybrid GRASP-evolutionary algorithm approach to golomb ruler search. In Yao, X., et al., eds.: Parallel Problem Solving From Nature VIII. Volume 3242 of Lecture Notes in Computer Science., Berlin, Springer-Verlag (2004) 481–490