Memetic Algorithms with Partial Lamarckism for 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. We consider the use of memetic algorithms (MAs) for solving this problem. A specialized local-improvement operator based on character removal and heuristic repairing plays a central role in the MA. The trade-off between the improvement achieved by this operator and its computational cost is analyzed. Empirical results indicate that strategies based on partial lamarckism (i.e., moderate use of the improvement operator) are slightly superior to full-lamarckism and no-lamarckism.

1 Introduction

The Shortest Common Supersequence Problem (SCSP) is a classical problem from the realm of string analysis. In essence, the SCSP consists of finding a minimal-length sequence S of symbols such that all strings in a certain set L are *embedded* in S (a more detailed description of the problem and the notion of embedding will be provided in next section). The SCSP provides a "clean" combinatorial optimization problem of great interest from the point of view of Theoretical Computer Science. In this sense, the SCSP has been studied in depth, and we now have accurate characterizations of its computational complexity. These characterizations range from the classical complexity paradigm (i.e., unidimensional complexity) to the more recent parameterized complexity paradigm (i.e., multidimensional complexity). We will survey some of these results in next section as well, but it can be anticipated that the SCSP is intrinsically hard [1–3] under many formulations and/or restrictions.

These hardness results would not be critical were the SCSP a mere academic problem. However, the SCSP turns out to be also a very important problem from an applied standpoint: it has applications in planning, data compression, and bioinformatics among other fields [4–6]. Thus, the practical impossibility of utilizing exact approaches for tackling this problem in general motivates attention be re-directed to heuristic approaches. Such heuristic approaches are aimed to producing *probably-* (yet not *provably-*) optimal solutions to the SCSP. Examples of such heuristics are the Majority Merge (MM) algorithm, and related variants [7], or the Alphabet-Leftmost (AL) algorithm [8]. More sophisticated heuristics have been also proposed, for instance, evolutionary algorithms (EAs) [7, 9].

This work will follow the way paved by previous EA approaches to this problem. To be precise, the use of memetic algorithms (MAs) will be considered. The main feature of this MA is the utilization of a twofold local-improvement strategy: on one hand, a repair mechanism is used to restore feasibility of solutions, shortening them if possible; on the other hand, an iterated local-search strategy is used to further improve solution quality. The computational impact of this latter component will be here analyzed, and confronted with the quality improvement attainable.

2 The Shortest Common Supersequence Problem

First of all, let us define the notion of supersequence. 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:

$$s \succ r \triangleq \begin{cases} \texttt{TRUE} \text{ if } r = \epsilon \\ \texttt{FALSE} \text{ if } r \neq \epsilon \text{ and } s = \epsilon \\ s' \succ r' \text{ if } r = \alpha r' \text{ and } s = \alpha s',\ \alpha \in \Sigma \\ s' \succ r \text{ if } r = \alpha r' \text{ and } s = \beta s' \text{ and } \alpha \neq \beta,\ \alpha, \beta \in \Sigma \end{cases} \tag{1}$$

Plainly, the definition above 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). We can now formally define the decisional version of the SCSP:

Shortest Common Supersequence Problem
Instance: A set L of m strings $\{s_1, \cdots, s_m\}$, $s_i \in \Sigma^*$ (where Σ is a certain alphabet), and a positive integer k.
Question: Does there exist a string $s \in \Sigma^*$, $|s| \leqslant k$, such that $s \succ s_i$ for all $s_i \in L$?

Obviously, associated with this decisional problem, we have its optimization version in which the smallest k is sought such that the corresponding instance is a yes-instance. Let us now consider the computational complexity of the SCSP.

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 [5], or when the alphabet size $|\Sigma|$ is two [2]. At any rate, it must be noted that NP-hard results are usually over-stressed; in fact, the NP-characterization is a worst-case scenario, and such worst cases may be unlikely (for example, 3-SAT is NP-hard, yet most instances are easily solvable; only those located at the phase transition between satisfiability and non-satisfiability are hard to solve). A more sensible characterization of hardness is required in order to deal with these issues, and parameterized complexity is the key.

Parameterized complexity [10] approaches problems from a multidimensional perspective, realizing its internal structure, and isolating some *parameters*. If hardness (that is, non-polynomial behavior) can be isolated within these parameters, the problem can be *efficiently* solved for fixed values of them. Here, efficiently means in time $O(f(k)n^c)$, where k is the parameter value, n is the problem size, f is an arbitrary function of k only, and c is a constant independent of k and n. A paradigmatic example of this situation is provided by Vertex Cover: it is NP-hard in general, but it can be solved in time $O(1.271^k+n)$, where n is the number of vertices, and k is the maximum size of the vertex cover sought [11, 12]. 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 class $W[1]$ is the current measure of intractability.

Several parameterizations are possible for the SCSP. 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, and these can be exhaustively checked. However, this is not very useful in practice because $k \geqslant \max |s_i|$. If the number of strings m is used as a parameter, then SCS is $W[1]-$hard, and remains so even if $|\Sigma|$ is taken as another parameter [6], or is constant [3]. 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, the absence of FPT algorithms implies the absence of fully polynomial time approximation schemes (FPTAS) for the corresponding problem, that is, there does not exist an algorithm returning solutions within factor $1+\epsilon$ from the optimum in time which is polynomial in n and $1/\epsilon$.

3 Heuristics for the SCSP

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 the strings in L, and removing these symbols from the corresponding strings. More precisely:

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

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 [7]. 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.* [7] propose to use precisely this string length as weight, i.e., step 3 in the previous pseudocode would be modified to have $\nu(\alpha) \leftarrow \sum_{s_i=\alpha s_i'} |s_i'|$.

Another heuristic has been proposed by Rahmann [8] in the context of the application of the SCSP to a microarray production setting. This algorithm is termed Alphabet-Leftmost (AL), and its functioning can be described as follows:

```
Heuristic AL ($L = \{s_1 \cdots, s_m\}$, $\Pi = \langle \pi_1 \cdots \pi_{|\Sigma|} \rangle$)
let $s \leftarrow \epsilon$
let $i \leftarrow 1$
do
    if $\exists s_j \in L: \ s_j = \pi_i s_j'$ then
        for $s_j \in L, s_j = \pi_i s_j'$ do let $s_j \leftarrow s_j'$
        let $s \leftarrow s\pi_i$
    end if
    let $i \leftarrow (i \ MOD \ |\Sigma|) + 1$
until $\sum_{s_i \in L} |s_i| = 0$
return $s$
```

As it can be seen, AL takes as input the list of strings whose supersequence is sought, and a permutation of symbols in the alphabet. The algorithm then proceeds with successive repetitions of this pattern until the all strings in L are embedded. Obviously, unproductive steps (i.e., when the next symbol in row does not appear at the front of any string in L) are ignored. Such a simple algorithm can provide very good results for some SCSP instances.

4 Memetic Algorithms 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. 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. We have analyzed these three approaches elsewhere, and we have found that option (ii) provided better results than option (i) and (iii) (in this latter case, we considered an EA that used ideas from GRASP [13] as suggested in [14]). We will thus elaborate on this 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 function $\rho : \Sigma^* \times (\Sigma^*)^m \to \Sigma^*$ whose behavior can be described as follows:

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

where $L|_\alpha = \{s_1|_\alpha, \cdots, s_m|_\alpha\}$, and $s|_\alpha$ equals s' when $s = \alpha s'$, being s otherwise. As it can be seen, this repairing function not only completes s in order to have a valid supersequence, but also removes unproductive steps, as it is done in AL. Thus, it also serves the purpose of local improver to some extent. After this repairing, raw fitness (to be minimized) is simply computed as:

$$fitness\,(s, L) = \begin{cases} 0 & \text{if } \forall i : s_i = \epsilon \\ 1 + fitness(s', L|_\alpha) & \text{if } \exists i : s_i \neq \epsilon \text{ and } s = \alpha s' \end{cases} \tag{3}$$

As mentioned in Sect. 1, an additional local-improvement level is considered. To do so, we have considered the neighborhood define by the $\textsc{Delete}_k : \Sigma^* \times (\Sigma^*)^m \to \Sigma^*$ operation [8]. The functioning of this operation is as follows:

$$\textsc{Delete}_k\,(s, L) = \begin{cases} \rho(s', L) & \text{if } k = 1 \text{ and } s = \alpha s' \\ \alpha\textsc{Delete}_{k-1}(s', L|_\alpha) & \text{if } k > 1 \text{ and } s = \alpha s' \end{cases} \tag{4}$$

This operation thus removes the k-th symbol from s, and then submits it to the repair function so that all all strings in L can be embedded. Notice that the repairing function can actually find that the sequence is feasible, hence resulting in a reduction of length by 1 symbol. A full local-search scheme is defined by iterating this operation until no single deletion results in length reduction:

```
Heuristic LS ($s \in \Sigma^*$, $L = \{s_1 \cdots, s_m\}$)
  let $k \leftarrow 0$
  while $k < |s|$ do
    let $r \leftarrow \textsc{Delete}_k(s, L)$
    if $fitness(r) < fitness(s)$ then
      let $s \leftarrow r$
      let $k \leftarrow 0$
    else
      let $k \leftarrow k + 1$
    end if
  end while
  return $s$
```

The application of this LS operator has a computational cost that we measure as the number of partial evaluations in step 3 above. More precisely, since the application of the repairing function starts at position k, we compute each application of $\textsc{Delete}_k$ to s as $(|r| - k)/|r|$ fitness evaluations. This is accumulated during the run of the MA to have a more sensible estimation of the search cost.

5 Experimental Results

The experiments have been done with a steady-state MA ($popsize = 100$, $p_X = .9$, $p_m = 1/n$, $maxevals = 100,000$), using binary tournament selection, uniform crossover, and random-substitution mutation. In order to analyze the impact of local search, the LS operation is not always applied, but randomly with probability p. The values $p \in \{0, 0.01, 0.1, 0.5, 1\}$ have been considered. We denote by $\text{MA}^{x\%}$ the use of $p = x/100$. Notice that $\text{MA}^{0\%}$ would then be a plain repair-based EA.

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. 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.

Secondly, a more realistic benchmark consisting of strings with a common source has been considered. A DNA sequence from a SARS coronavirus strain has been retrieved from a genomic database[1], 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.

First of all, the results for the random strings are shown in Table 1. All MAs perform notably better than AL. The results for MM (not shown) are similar to those of AL (more precisely, they are between 2.5% and 10% better, still far worse than the MAs). Regarding the different MAs, performance differences tend to be higher for increasing alphabet sizes. In general, MAs with $p > 0$ are better than $\text{MA}^{0\%}$ (the differences are statistically significant according to a Wilcoxon ranksum test [15] in above 90% of the problem instances). $\text{MA}^{1\%}$ provides somewhat better results, although the improvement with respect to the other MAs ($p > 0$) is only significant in less than 20% of the problem instances.

The results for the strings from the SARS DNA sequence are shown in Table 2. Again, AL performs quite poorly here. Unlike the previous set of instances, MM (not shown) does perform notably better than AL. Actually, it matches the performance of $\text{MA}^{0\%}$ for low gap probability (10% and 15%), and yields an average 227.8 for the larger gap probability. In this latter problem instance, the

[1] `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 AL are averaged over all permutations of the alphabet (or a maximum 100,000 permutations for $|\Sigma| = 16$), and the results of the EAs are averaged over 30 runs. In all cases, the results are further averaged over five different problem instances.

	AL		$\text{MA}^{0\%}$		$\text{MA}^{1\%}$	
$\|\Sigma\|$	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
2	121.4	123.4 ± 2.0	111.2	112.6 ± 0.8	110.4	112.8 ± 1.0
4	183.0	191.2 ± 4.7	151.6	155.2 ± 1.9	149.4	152.7 ± 1.7
8	252.2	276.8 ± 6.4	205.4	213.5 ± 4.0	201.8	207.3 ± 2.2
16	320.6	352.9 ± 7.4	267.0	281.8 ± 5.9	266.2	274.2 ± 3.0
	$\text{MA}^{10\%}$		$\text{MA}^{50\%}$		$\text{MA}^{100\%}$	
$\|\Sigma\|$	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
2	111.6	113.1 ± 0.8	111.4	113.2 ± 0.8	111.2	113.1 ± 0.8
4	149.4	153.5 ± 1.5	150.0	153.3 ± 1.4	149.2	153.3 ± 1.6
8	202.0	207.9 ± 2.2	204.0	208.2 ± 2.0	203.0	208.2 ± 2.1
16	266.6	274.7 ± 3.0	268.4	275.0 ± 2.8	267.2	275.0 ± 3.2

Table 2. Results of the different heuristics on the strings from the SARS DNA sequence. The results of AL are averaged over all permutations of the alphabet (or a maximum 100,000 permutations for $|\Sigma| = 16$), and the results of the EAs are averaged over 30 runs.

	AL		$\text{MA}^{0\%}$		$\text{MA}^{1\%}$	
gap%	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
10%	307	315.2 ± 6.8	158	158.0 ± 0.0	158	158.0 ± 0.0
15%	293	304.3 ± 8.8	158	158.0 ± 0.0	158	159.0 ± 2.8
20%	274	288.3 ± 8.6	165	180.8 ± 15.7	159	177.0 ± 9.3
	$\text{MA}^{10\%}$		$\text{MA}^{50\%}$		$\text{MA}^{100\%}$	
gap%	best	mean ± std.dev.	best	mean ± std.dev.	best	mean ± std.dev.
10%	158	158.0 ± 0.0	158	158.0 ± 0.0	158	158.0 ± 0.0
15%	158	159.8 ± 3.7	158	159.8 ± 3.0	158	159.1 ± 2.1
20%	163	179.4 ± 9.2	159	178.1 ± 9.9	161	176.5 ± 9.0

MAs with $p > 0$ seem to perform marginally better. $\text{MA}^{100\%}$ and $\text{MA}^{1\%}$ provide the best and second best mean results (no statistical difference between them). A Wilcoxon ranksum test indicates that the difference with respect to $\text{MA}^{0\%}$ is significant (at the standard 5% significance).

6 Conclusions

We have studied the deployment of MAs on the SCSP. The main goal has been to determine the way that local search affects the global performance of the algorithm. The experimental results seem to indicate that performance differences are small but significant with respect to a plain repair-based EA (i.e., no local search). Using partial lamarckism ($0 < p < 1$) provides in some problem instances better results, and does not seem to be harmful on any of the remaining instances. Hence, it can offer the best tradeoff between quality improvement and computational cost. Future work will be directed to confirm these results on other neighborhood structures for local search. In this sense, alternatives based on symbol insertions or symbol swaps can be considered [8].

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

References

1. Bodlaender, H., Downey, R., Fellows, M., Wareham, H.: The parameterized complexity of sequence alignment and consensus. Theoretical Computer Science **147** (1994) 31–54
2. Middendorf, M.: More on the complexity of common superstring and supersequence problems. Theoretical Computer Science **125** (1994) 205–228
3. 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
4. Foulser, D., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence **57** (1992) 143–181
5. Timkovsky, V.: Complexity of common subsequence and supersequence problems and related problems. Cybernetics **25** (1990) 565–580
6. Hallet, M.: An integrated complexity analysis of problems from computational biology. PhD thesis, University of Victoria (1996)
7. Branke, J., Middendorf, M., Schneider, F.: Improved heuristics and a genetic algorithm for finding short supersequences. OR-Spektrum **20** (1998) 39–45
8. Rahmann, S.: The shortest common supersequence problem in a microarray production setting. Bioinformatics **19** (2003) ii156–ii161
9. Branke, J., Middendorf, M.: Searching for shortest common supersequences by means of a heuristic based genetic algorithm. In: Proceedings of the Second Nordic Workshop on Genetic Algorithms and their Applications, Finnish Artificial Intelligence Society (1996) 105–114
10. Downey, R., Fellows, M.: Parameterized Complexity. Springer-Verlag (1998)
11. Chen, J., Kanj, I., Jia, W.: Vertex cover: further observations and further improvements. In: Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science. Number 1665 in Lecture Notes in Computer Science, Berlin Heidelberg, Springer-Verlag (1999) 313–324
12. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Information Processing Letters **73** (2000) 125–129
13. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization **6** (1995) 109–133
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
15. Lehmann, E.: Nonparametric Statistical Methods Based on Ranks. McGraw-Hill, New York NY (1975)