A Probabilistic Beam Search Approach to the Shortest Common Supersequence Problem*

[1]Christian Blum [2]Carlos Cotta [2]Antonio J. Fernández [2]Francisco Gallardo

[1]Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona 1–3, Ω building, Campus Nord, E-08034 Barcelona (Spain)
cblum@lsi.upc.edu

[2]Dept. Lenguajes y Ciencias de la Computación
ETSI Informática, Universidad de Málaga
Campus de Teatinos, E-29071 Málaga (Spain)
{ccottap,afdez,pepeg}@lcc.uma.es

Abstract

The Shortest Common Supersequence Problem (SCSP) is a well-known hard combinatorial optimization problem that formalizes many real world problems. This paper presents a novel randomized search strategy, called probabilistic beam search (PBS), based on the hybridization between beam search and greedy constructive heuristics. PBS is competitive (and sometimes better than) previous state-of-the-art algorithms for solving the SCSP. The paper describes PBS and provides an experimental analysis (including comparisons with previous approaches) that demonstrate its usefulness.

1 Introduction

The Shortest Common Supersequence Problem (SCSP) is a very well-known problem in the area of string analysis. Basically, the SCSP consists of finding a minimal-length sequence s of symbols from a certain alphabet, such that all strings in a given set L can be *embedded* in s. The resulting combinatorial problem is enormously interesting for several reasons. Firstly, the SCSP constitutes a formalization of different real-world problems. For example, it has many implications in bioinformatics [10]: it is a problem with a close relationship to multiple sequence alignment [16], and to probe synthesis during microarray production [15]. This does not exhaust the practical usefulness of the SCSP though, since it also has applications in planning [8] and data compression [17], among other fields.

Another reason the SCSP has attracted interest lies in its "cleanliness", that is, it is an abstract formulation of different real-world problems that can nevertheless be studied from a theoretical point of view in a context-independent way. Indeed, theoretical computer

*Christian Blum acknowledges support from the Spanish Ministry of Education and Science under the project CICYT TIN-2005-08818-C04-01 (OPLINK), and from the "Ramón y Cajal" program of the Spanish Ministry of Science and Technology of which he is a post-doctoral research fellow.

scientists have analyzed in depth the problem, and we now have accurate characterizations of its computational complexity. These characterizations range from the classical complexity paradigm to the more recent parameterized complexity paradigm. We will survey some of these results in next section as well, but it can be anticipated that the SCSP is intrinsically hard [1, 11, 14] under many formulations and/or restrictions.

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. Good examples of such heuristics are the Majority Merge (MM) algorithm, and related variants [3], based on greedy construction strategies. More sophisticated heuristics have been also proposed, for instance, evolutionary algorithms (EAs) [3, 2, 5, 6]. In this work, we present a novel randomized search strategy (or metaheuristic) to tackle the SCSP termed probabilistic beam search (PBS). As the name indicates, this strategy is based in the framework of beam search, but also borrows some heuristic ideas from the greedy constructive heuristics mentioned before. In the following we will show that this strategy can satisfactorily compete in the SCSP arena, outperforming previous state-of-the-art approaches. As a first step, the next section will describe the SCSP in more detail.

2 The Shortest Common Supersequence Problem

First of all, let us introduce some notation that we use in the following. We write $|s|$ for the length of string s ($|s(1)s(2)\dots s(n)| = n$, where $s(j) \in \Sigma$ is the element at the j-th position of s) 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. Deleting symbol α from the front of string s is denoted by $s|_\alpha$, and is defined as s' when $s = \alpha s'$, or s otherwise. We also use the $|$ symbol to delete a symbol from the front of a set of strings: $\{s_1, \cdots, s_m\}|_\alpha = \{s_1|_\alpha, \cdots, s_m|_\alpha\}$. 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:

$$\begin{array}{rcll} s \succ \epsilon & \triangleq & \text{True} & \\ \epsilon \succ r & \triangleq & \text{False}, & \text{if } r \neq \epsilon \\ \alpha s \succ \alpha r & \triangleq & s \succ r & \\ \alpha s \succ \beta r & \triangleq & s \succ \beta r, & \text{if } \alpha \neq \beta \end{array} \tag{1}$$

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 ($s \succ s_i, \forall s_i \in L$ and $|s|$ is minimal). 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. For example, it is NP$-$hard in general when all s_i have length two [17], or when the alphabet

size $|\Sigma|$ is two [11]. In principle, these NP−hardness results would have to be approached with caution, since they merely represent a worst case scenario. In this sense, a more sensible characterization of the hardness of the SCSP is provided by the framework of parameterized complexity [7]. This is done by approaching the problem 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*[1] solved for fixed values of them. This is the case for several NP−hard problems such as VERTEX COVER [4, 12]; the term *fixed-parameter tractable* (FPT) is used to denote these problems. Non-FPT problems 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 SCSP is $W[1]$−hard, and remains so even if $|\Sigma|$ is taken as another parameter [10], or is constant [14]. 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.

3 Majority Merge Heuristics for the SCSP

The hardness results mentioned previously motivate the utilization of heuristics 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 ($L = \{s_1, \cdots, s_m\}$)
1: **let** $s \leftarrow \epsilon$
2: **do**
3: **for** $\alpha \in \Sigma$ **do let** $\nu(\alpha \mid s) \leftarrow \sum_{s_i \in L, s_i = \alpha s_i'} 1$
4: **let** $\beta \leftarrow \text{argmax}\{\nu(\alpha \mid s) \mid \alpha \in \Sigma\}$
5: **let** $L \leftarrow L|_\beta$
6: **let** $s \leftarrow s\beta$
7: **until** $\sum_{s_i \in L} |s_i| = 0$
8: **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 [3]. 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 on the length of the string

[1] 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.

in whose front is located. Branke *et al.* [3] propose to use precisely this string length as weight, i.e., step 3 in the previous pseudocode would be modified to have

$$\nu(\alpha \mid s) \leftarrow \sum_{s_i \in L, s_i = \alpha s'_i} |s'_i| \tag{2}$$

This modified heuristic is termed Weighted Majority Merge (WMM), and its empirical evaluation indicates it can outperform MM on some problem instances in which there is no structure, or the structure is deceptive [3, 5].

In this work we also consider look-ahead versions of the WMM heuristic. For that purpose we use the notation LA-WMM(l), where $l > 0$ is a parameter that indicates the size (or depth) of the look-ahead. For example, LA-WMM(0) denotes the standard WMM heuristic, whereas LA-WMM(1) is obtained by choosing at each construction step the symbol that corresponds to the first symbol in the best possible sequence of two WMM construction steps. The value of a sequence of two construction steps is obtained by summing the two corresponding WMM weights (see Equation 2). In the following we will refer to these look-head values as the LA-WMM(l) weights.

4 Probabilistic Beam Search for the SCSP

In the following we present a probabilistic beam search (PBS) approach for the SCSP. This algorithm is based on the WMM heuristic outlined before. Beam search is a classical tree search method that was introduced in the context of scheduling [13]. The central idea behind beam search is to allow the extension of partial solutions in more than one way. The version of beam search that we implemented—see algorithm PBS below—works as follows: At each step of the algorithm is given a set B of partial solutions which is called the *beam*. At the start of the algorithm B only contains the empty partial solution ϵ (that is, $B = \{\epsilon\}$). Let C denote the set of all possible children of the partial solutions in B. Note that a child of a string s is obtained by appending one of the symbols from Σ to it. At each step, k_{ext} different (partial) solutions from C are selected; each selection step is either performed probabilistically or deterministically. A chosen (partial) solution is either stored in set B_{compl} in case it is a complete solution, or in the new beam B otherwise. At the end of each construction step the new beam B is reduced in case it contains more than k_{bw} (called the *beam width*) partial solutions. This is done by evaluating the partial solutions in B by means of a lower bound $\text{LB}(\cdot)$, and by subsequently selecting the k_{bw} partial solutions with the smallest lower bound values.

Algorithm PBS($k_{\text{ext}}, k_{\text{bw}}, s_{\text{bsf}}, d$)
1: **let** $B_{\text{compl}} = \emptyset$
2: **let** $B = \{\epsilon\}$
3: **while** $B \neq \emptyset$
4: **let** $C \leftarrow$ Children_of(B)
5: **let** $B \leftarrow \emptyset$
6: **for** $k = 1, \ldots, k_{\text{ext}}$ **do**
7: **let** $s^t \leftarrow$ Choose_From(C, d)
8: **if** $\text{LB}(s^t) = |s^t|$ **then**
9: **let** $B_{\text{compl}} \leftarrow B_{\text{compl}} \cup \{s^t\}$

```
10:            if |s^t| < |s_bsf| then s_bsf ← s^t endif
11:         else
12:            if LB(s^t) ≤ |s_bsf| then B ← B ∪ {s^t} endif
13:         end if
14:         let C ← C \ {s^t}
15:      end for
16:      let B ← REDUCE(B,k_bw)
17:   end while
18:   return argmin {|s| |s ∈ B_compl}
```

In the following we explain the functions of algorithm PBS in more detail. First of all, let us define the following function that will be useful to calculate lower bounds of partial solutions:

$$\begin{array}{rcll} s \gg \epsilon & \triangleq & (\epsilon, \epsilon) & \\ \epsilon \gg r & \triangleq & (\epsilon, r), & \text{if } r \neq \epsilon \\ \alpha s \gg \alpha r & \triangleq & (\alpha r^e, r^r), & \text{where } (r^e, r^r) = s \gg r \\ \alpha s \gg \beta r & \triangleq & s \gg \beta r, & \text{if } \alpha \neq \beta \end{array} \tag{3}$$

Intuitively, $s \gg r = (r^e, r^r)$ if r^e is the longest initial segment of string r embedded by s and r^r is the remaining part of r not embedded by s (i.e., $r = r^e r^r$). Note that $s \succ r \iff s \gg r = (r, \epsilon)$.

Function CHILDREN_OF(B) produces the set C of all possible children of the partial solutions in B. Note that, given a partial solution s^t, at most $|\Sigma|$ children can be generated by appending each of the symbols from Σ to s^t. Children with unproductive characters (i.e., not contributing to embedding any string in L) are not added to C.

Another important function of algorithm PBS is CHOOSE_FROM(C, d). Upon invocation, this function returns one of the partial solutions from set C. This is done as follows. First, we calculate for each $s^t \in C$ a heuristic value $\eta(s^t)$ as follows:

$$\eta(s^t) \leftarrow \left(\sum_{i=1}^{|s^t|} \nu^r \left(s^t(i) \mid s^t(1) s^t(2) \ldots s^t(i-1) \right) \right)^{-1}, \tag{4}$$

where $\nu^r(\alpha \mid s)$ is the rank of the weight $\nu(\alpha \mid s)$ which the LA-WMM(l) heuristic assigns to the extension α of string s (see Section 3). The rank of extending string s by symbol α is obtained by sorting all possible extensions of string s with respect to their LA-WMM(l) weights in descending order. Note that the sum shown in Equation 4 is the sum of the ranks of the LA-WMM(l) weights that are used for constructing the partial solution s^t. For example, in case s^t can be constructed by always appending the symbol suggested by the LA-WMM(l) heuristic, the heuristic value of s^t is $\eta(s^t) = \left(\sum_{i=1}^{|s^t|} 1 \right)^{-1} = (|s^t|)^{-1}$. This way of defining the heuristic values has the effect that partial solutions obtained by mostly following the suggestions of the LA-WMM(l) heuristic have a greater heuristic value than others. Given the heuristic values we can define the probability of a (partial) solution s^t from C to be chosen in function CHOOSE_FROM(C, d):

$$\mathbf{p}(s^t) \leftarrow \frac{\eta(s^t)}{\sum_{s^l \in C} \eta(s^l)} \tag{5}$$

However, instead of always choosing a partial solution $s^t \in C$ probabilistically, we employ the following mixed strategy. First, a random number $r \in [0,1]$ is drawn. If $r < d$ (where $d \in [0,1]$ is a parameter of the algorithm), the partial solution s^* to be returned by function CHOOSE_FROM(C,d) is selected such that $s^* \leftarrow \text{argmax}\{\mathbf{p}(s^t) \mid s^t \in C\}$. Otherwise, a partial solution is chosen by roulette-wheel-selection using the probabilities defined in Equation 5.

Finally, the lower bound $\text{LB}(s^t)$ of a partial solution s^t is calculated as follows: First, we calculate the set of remaining strings in L not embedded by s^t as follows:

$$R(s^t) = \{r_i \mid (s_i^e, r_i) = s^t \gg s_i, s_i \in L\} \tag{6}$$

Let $M(\alpha, R(s^t))$ be the maximum number of occurrences of symbol α in any string in $R(s^t)$:

$$M(\alpha, R(s^t)) = \max\{r_i \trianglerighteq \alpha \mid r_i \in R(s^t)\} \tag{7}$$

Clearly, every common supersequence for the remaining strings must contain at least $M(\alpha, R(s^t))$ copies of the symbol α. Thus a lower bound is obtained by summing the length of the partial solution s^t and the maximum number of occurrences of each symbol of the alphabet in any string in $R(s^t)$:

$$|s^t| + \sum_{\alpha \in \Sigma} M(\alpha, R(s^t)) \tag{8}$$

Note that we use algorithm PBS in a multi-start fashion, that is, given a CPU time limit we apply algorithm PBS over and over again until the CPU limit is reached. The best solution found, denoted by s_{bsf}, is recorded. In fact, this solution is one of the input parameters of algorithm PBS. It is used to exclude partial solutions whose lower bound value exceeds $|s_{\text{bsf}}|$ from further consideration.

5 Experimental Evaluation

We implemented our algorithm in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with an AMD64X2 4400 processor and 4 Gb of memory.

Two different sets of benchmark instances have been used in the experimentation. The first one—henceforth referred to as SET1—is composed of random strings with different lengths. To be precise, each instance is composed of eight strings, four of them of length 40, and the other four of length 80. Each of these strings is randomly generated, using an alphabet Σ. The benchmark set consists of 5 classes of each 5 instances characterized by different alphabet sizes, namely $|\Sigma| = 2$, 4, 8, 16, and 24. Accordingly, the benchmark set consists of 25 different problem instances. The same instances were used for experimentation, for example, in [5].

A second set of instances is composed of strings with a common source. To be precise, we have considered strings obtained from molecular sequences. The sequences considered comprise both DNA sequences ($|\Sigma| = 4$) and protein sequences ($|\Sigma| = 20$). In the first case, we have taken two DNA sequences of the SARS coronavirus from a genomic database[2]; these sequences are 158 and 1269 nucleotides long. As to the protein sequences, we have considered three of them, extracted from Swiss-Prot[3]:

[2] `http://gel.ym.edu.tw/sars/genomes.html`

[3] `http://www.expasy.org/sprot/`

- *Oxytocin*: quite important in pregnant women, this protein causes contraction of the smooth muscle of the uterus and of the mammary gland. The sequence is 125-aminoacid long.
- *p53*:this protein is involved in the cell cycle, and acts as tumor suppressor in many tumor types; the sequence is 393-aminoacid long.
- *Estrogen*: involved in the regulatin of eukaryotc gene expression, this protein affects cellular proliferation and differentiation; the sequence is 595-aminoacid long.

In all cases, problem instances are generated by generating strings from the target sequence by removing symbols from the latter with probability $p\%$. In our experiments, problem instances comprise 10 strings, and $p \in \{10\%, 15\%, 20\%\}$.

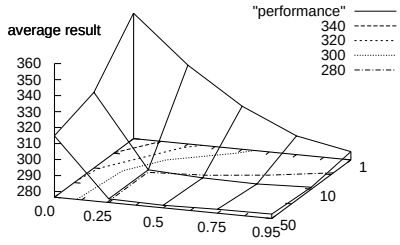
5.1 Algorithm tuning

First we wanted to find reasonable settings for the parameters of PBS. Remember that PBS has 4 parameters:

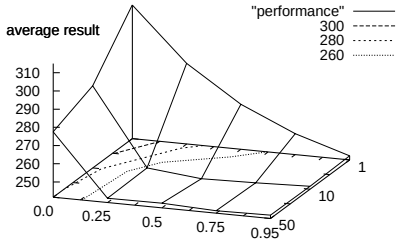
1. k_{bw} is the beam width;
2. k_{ext} is the number of children to be chosen from set C at each step;
3. d is the parameter that controls the extent to which the choice of children from C is performed deterministically. If $d = 1.0$, this choice is always done deterministically, whereas when $d = 0.0$ the choice is always done by rhoulette-wheel-selection;
4. Finally, l is the depth of the look-ahead function, that is, the parameter in LA-WMM(l) (see Section 3).

In order to reduce the set of parameters to be considered for tuning we decided beforehand to set $k_{\text{ext}} = 2 \cdot k_{\text{bw}}$. In preliminary experiments we found this setting to be reasonable. Concerning the remaining parameters we tested the following settings: $k_{\text{bw}} \in \{1, 10, 50\}$, $d \in \{0.0, 0.25, 0.5, 0.75, 0.95\}$, and $l \in \{0, 1, 2, 3\}$. First we studied the relation between parameters k_{bw} and d, fixing parameter l to the maximum value 3 (that is, $l = 3$). We applied PBS with each combination of parameter values 5 times for 500 CPU seconds to each of the problem instances of SET1. This provided us with 25 results for each instance class (as characterized by the alphabet size). The averaged results for each instance class are shown in the graphics of Figure 1. The results show that, in general, PBS needs some determinism in extension of partial solutions ($d > 0.0$), as well as a beam width greater than 1 ($d > 1$). However, in particular for the problem instances with a smaller alphabet size, the determinism should not be too high and the beam width should not be too big. Therefore, we decided for the settings $d = 0.5$ and $k_{\text{bw}} = 10$ for all further experiments.

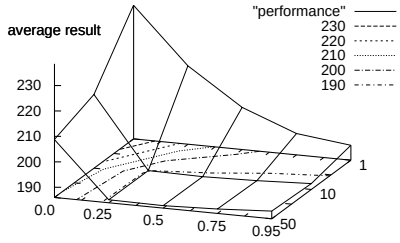
Finally we performed experiments to decide for the setting of l, that is, the parameter of the look-ahead mechanism. We applied PBS with the four different settings of l ($l \in \{0, 1, 2, 3\}$ 5 times for 500 CPU seconds to each of the problem instances of SET1. This provides us with 25 results for each instance class. The averaged results for each instance class are shown in the graphics of Figure 2. The results show that, in general, the setting of $l = 3$ is best. Especially when the alphabet size is rather large, the performance of PBS is better the higher l is. Only for $\Sigma = 2$, the setting of l does not play much of a role. Therefore, we decided for the setting $l = 3$ for all further experiments.



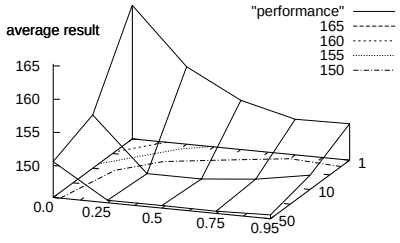
(a) $\Sigma = 24$



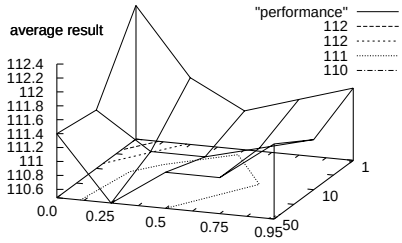
(b) $\Sigma = 16$



(c) $\Sigma = 8$



(d) $\Sigma = 4$



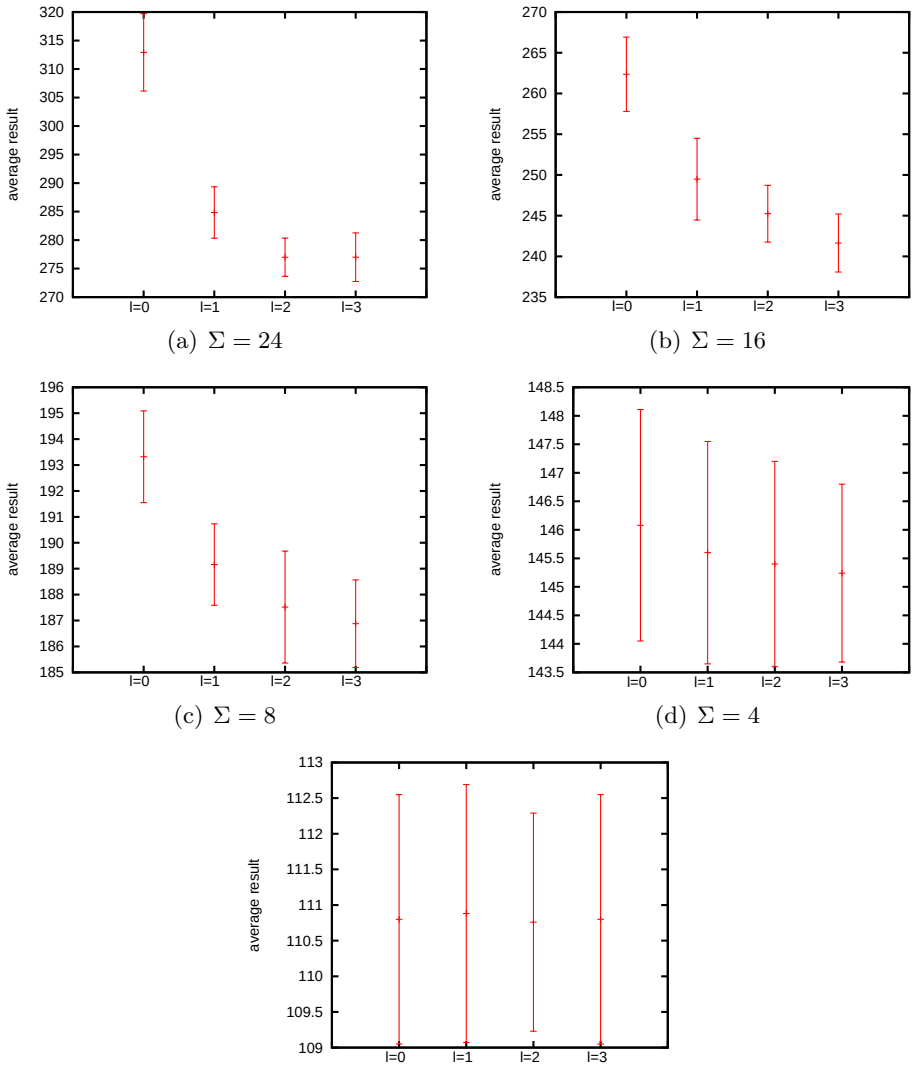
(e) $\Sigma = 2$

Figure 1: The z-axis of each graphic shows the average performance of PBS with the parameter settings as specified by the x-axis (parameter d) and the y-axis (parameter k_{bw}).

5.2 Final Experimental Evaluation

We compare the results of PBS to 3 different algorithms: MM refers to a multi-start version of the MM heuristic. This can be done as in case of ties during the solution construction they are broken randomly. Furthermore, WMM refers to a multi-start version of the WMM heuristic, and Hybrid MA-BS refers to an algorithm that is a hybrid between beam search and a memetic algorithm. Note that Hybrid MA-BS is a current state-of-the-art technique for the SCSP. The results for all three techniques are taken from [9]. The stopping criterion of MM, WMM, and Hybrid MA-BS was 600 CPU time seconds on a Pentium IV PC with 2400 MHz and 512 Mb of memory. This corresponds roughly to the 350 CPU time seconds that we allowed on our machine for PBS.

First, we present the results of PBS for the instances of SET1 in numerical form in Table 1.



(a) $\Sigma = 24$ (b) $\Sigma = 16$

(c) $\Sigma = 8$ (d) $\Sigma = 4$

(e) $\Sigma = 2$

Figure 2: The y-axis of each graphic shows the average performance (and its standard deviation) of PBS with the parameter setting of l as specified by the x-axis.

The results show that PBS is always better than the basic greedy heuristics. With respect to the more sophisticated MA-BS algorithm, the results of PBS are roughly equivalent for $|\Sigma| = 2$. In the remaining instances, PBS improves significantly over the results of Hybrid MA-BS. Even the average performance of PBS is always better than the best performance of Hybrid MA-BS.

As to the biological sequences, the results are shown in Table 2. Again, PBS can be seen to be notoriously better than the greedy algorithms. With respect to MA-BS, PBS is capable of performing at the same level in most instances, systematically finding the optimal solutions. Only in the largest problem instances PBS starts to suffer from the curse of dimensionality. Notice nevertheless that PBS has still room for improvement. For example, using a larger beam width $k_{bw} = 100$ (instead of $k_{bw} = 10$), the results for the two harder

Table 1: Results for the instances of SET1.

	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

	Hybrid MA-BS			PBS		
$\|\Sigma\|$	best	mean $\pm$ σ	i.%	best	mean $\pm$ σ	i.%
2	110.6	110.7 $\pm$ 0.0	1.2	110.8	110.9 $\pm$ 1.7	1.0
4	145.6	146.4 $\pm$ 0.5	4.6	144.8	145.4 $\pm$ 1.5	5.2
8	191.6	192.6 $\pm$ 1.4	9.9	186.4	187.2 $\pm$ 1.7	12.4
16	242.8	244.0 $\pm$ 1.0	14.7	240.4	241.9 $\pm$ 3.4	15.4
24	280.2	281.2 $\pm$ 0.8	15.8	276.4	277.9 $\pm$ 4.0	16.8

SARS DNA instances are notably improved: for 15% gap, the mean result is 1269$\pm$0.0 (i.e., systematically finding the optimal solution); for 20% gap, the mean result is 1483$\pm$143.1 (best result = 1294) which is much closer to optimal. Further fine-tuning of the parameters may produce even better results.

6 Conclusions and future work

We have introduced PBS, a novel metaheuristic that blends ideas from beam search and randomized greedy heuristics. Though relatively simple, and with just four parameters, PBS has been shown to be competitive with a much more complex hybrid metaheuristic for the SCSP that combines beam search and memetic algorithms. Furthermore, PBS has clearly outperformed this latter algorithm in one set of instances. In all cases, PBS has been also shown to be superior to two popular greedy heuristics for the SCSP.

The scalability of PBS is one of the features that deserves further exploration. As indicated by current results, an adequate parameterization of the algorithm can lead to improved results. The underlying greedy heuristic using within PBS, or the probabilistic choosing procedure can be also adjusted. The possibilities are manifold, and work is currently underway in this direction. An additional line of research is the hybridization of PBS with memetic algorithms. A plethora of models are possible in this sense, and using the same algorithmic template of the MA-BA hybrid would be a natural first step.

References

[1] H.L. Bodlaender, R.G. Downey, M.R. Fellows, and H.T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1–2):31–54, 1994.

[2] J. Branke and M. Middendorf. Searching for shortest common supersequences by means of a heuristic based genetic algorithm. In *Proceedings of the Second Nordic Workshop on*

Table 2: Results of the different algorithms for the biological sequences.

158-nucleotide SARS sequence

	MM		WMM		Hybrid MA-BS		PBS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	158	158.0 $\pm$ 0.0	158	158.0 $\pm$ 0.0	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	158	158.0 $\pm$ 0.0	158	158.0 $\pm$ 0.0
20%	228	229.6 $\pm$ 1.8	266	266.0 $\pm$ 0.0	158	158.0 $\pm$ 0.0	158	158.0 $\pm$ 0.0

1269-nucleotide SARS sequence

	MM		WMM		Hybrid MA-BS		PBS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	1970	2039.9 $\pm$ 32.9	2455	2455.0 $\pm$ 0.0	1269	1269.0 $\pm$ 0.0	1269	1269.0 $\pm$ 0.0
15%	2151	2236.4 $\pm$ 30.4	2346	2346.0 $\pm$ 0.0	1269	1269.0 $\pm$ 0.0	1269	1303.8 $\pm$ 36.6
20%	2163	2180.2 $\pm$ 13.9	2207	2207.0 $\pm$ 0.0	1269	1269.0 $\pm$ 0.0	1571	1753.2 $\pm$ 61.0

125-aminoacid Oxytocyn sequence

	MM		WMM		Hybrid MA-BS		PBS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	126	126.0 $\pm$ 0.0	126	126.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0
15%	126	126.0 $\pm$ 0.0	126	126.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0
20%	132	132.0 $\pm$ 0.0	227	227.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0	125	125.0 $\pm$ 0.0

393-aminoacid p53 sequence

	MM		WMM		Hybrid MA-BS		PBS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	393	393.0 $\pm$ 0.0	396	396.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0
15%	422	422.0 $\pm$ 0.0	832	832.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0
20%	612	677.1 $\pm$ 40.7	833	833.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0	393	393.0 $\pm$ 0.0

595-aminoacid Estrogen sequence

	MM		WMM		Hybrid MA-BS		PBS	
gap%	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ	best	mean $\pm$ σ
10%	628	628.0 $\pm$ 0.0	1156	1156.0 $\pm$ 0.0	595	595.0 $\pm$ 0.0	595	595.0 $\pm$ 0.0
15%	671	672.9 $\pm$ 2.0	1232	1242.1 $\pm$ 4.5	595	595.0 $\pm$ 0.0	595	595.0 $\pm$ 0.0
20%	1071	1190.3 $\pm$ 66.2	1324	1327.9 $\pm$ 4.6	595	595.0 $\pm$ 0.0	596	596.0 $\pm$ 0.0

Genetic Algorithms and their Applications, pages 105–114. Finnish Artificial Intelligence Society, 1996.

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

[4] J. Chen, I.A. Kanj, and W. Jia. 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, pages 313–324, Berlin Heidelberg, 1999. Springer-Verlag.

[5] C. Cotta. A comparison of evolutionary approaches to the shortest common supersequence problem. In J. Cabestany, A. Prieto, and D.F. Sandoval, editors, *Proceedings of the Eight International Workconference on Artificial Neural Networks*, volume 3512 of *Lecture Notes in Computer Science*, pages 50–58, Berlin, 2005. Springer-Verlag.

[6] C. Cotta. Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In J. Mira and J.R. Álvarez, editors, *Proceedings of the 1st International Work-conference on the Interplay between Natural and Artificial Computation*, number 3562 in Lecture Notes in Computer Science, pages 84–91, Berlin Heidelberg, 2005. Springer-Verlag.

[7] R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[8] D.E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57(2-3):143–181, 1992.

[9] J. E. Gallardo, C. Cotta, and A. J. Fernández. Hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 2006. in press.

[10] M.T. Hallet. *An integrated complexity analysis of problems from computational biology*. PhD thesis, University of Victoria, 1996.

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

[12] R Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

[13] P. S. Ow and T. E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:297–307, 1988.

[14] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(1):757–771, 2003.

[15] S. Rahmann. The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19(Suppl. 2):ii156–ii161, 2003.

[16] J.S. Sim and K. Park. The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, 1(1):111–117, 2003.

[17] V.G. Timkovsky. Complexity of common subsequence and supersequence problems and related problems. *Cybernetics*, 25:565–580, 1990.