1 Metaheuristics in Bioinformatics: DNA Sequencing and Reconstruction

C. COTTA, A. J. FERNÁNDEZ, J. E. GALLARDO, G. LUQUE, E. ALBA
Dpto. de Lenguajes y Ciencias de la Computación, E.T.S.I. Informática
Campus Teatinos 29071, Málaga, SPAIN
e-mail: {ccottap,afdez,pepeg,gabriel,eat}@lcc.uma.es

1.1 INTRODUCTION

In the past decades advances in fields of molecular biology and genomic technologies have led to a very important growth in the biological information generated by the scientific community. The needs of biologist to utilize, interpret, and analyze that large amount of data have increased the importance of the bioinformatics [1]. This area is an interdisciplinary field involving biology, computer science, mathematics, and statistics for achieving faster and better methods in that tasks.

Most of the bioinformatic tasks are formulated as hard combinatorial problems. Thus, is not viable to solve large instances of it using exact techniques such as branch and bound. As a consequence the use of metaheuristics and other approximate techniques is mandatory. In short, a metaheuristic [2, 3] can be defined as a top-level general strategy which guides other heuristics to search for good solutions. Up to now there is no commonly accepted definition for the term metaheuristic. It is just in the last few years that some researchers in the field tried to propose a definition. Some fundamental characteristics:

- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Metaheuristic algorithms are usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.

- The basic concepts of metaheuristics permit an abstract level description.
- Metaheuristics are not problem-specific.
- Metaheuristics usually allow an easy parallel implementation.
- Metaheuristics must make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

The main advantages of using metaheuristics to solve bioinformatics tasks are the following:

- Problems of bioinformatics seldom need the optimal solution. In fact, they require robust, fast and near-optimal solutions.
- Data obtained from laboratories inherently involve errors. Metaheuristics, due to their non-deterministics process, are more tolerant in these cases than deterministic ones.
- Several tasks in bioinformatics involve the optimization of different objectives, thereby making the application of (population based) metaheuristics more natural and appropriate.

In this chapter, we first present a brief survey about metaheuristic techniques and main bioinformatic tasks. Later, we describe in more detail two important problems in the area of sequence analysis: the DNA fragment assembly and the shortest common supersequence problem. We use them to exemplify how metaheuristics can be used to solve difficult bioinformatic tasks.

1.2 METAHEURISTICS AND BIOINFORMATICS

In this section we present some background information about metaheuristics and problems of bioinformatics.

1.2.1 Metaheuristics

As we said before, a inxmetaheuristic [2, 3] can be defined as a top-level general strategy which guides other heuristics to search for good solutions. There are different ways to classify and describe metaheuristic algorithms. One of them classifies them depending on the number of solutions: population based (a set of solutions) and trajectory based (work with a single solution). The former starts with a single initial solution. At each step of the search the current solution is replaced by another (often the best) solution found in its neighborhood. Very often, they allow to find a local optimal solution, and so are called exploitation-oriented methods. On the other hand, the latter make use of a randomly generated population of solutions. The initial population is enhanced through a natural evolution process. At each generation of the process, the whole population or a part of the population is replaced by newly generated individuals (often the best ones). Population based methods are often called exploration-oriented methods. In next paragraph we discuss the features of the most important metaheuristics.

Trajectory based metaheuristics:

- **Simulated Annealing** (SA) [4] is a stochastic search method in which at each step, the current solution is replaced by another one that improves the objective function, randomly selected from the neighborhood. SA uses a control parameter, called temperature, to determine the probability of accepting non-improving solutions. The objective is to escape from local optima, and so to delay the convergence. The temperature is gradually decreased according to a cooling schedule such that few non-improving solutions are accepted at the end of the search.

- **Tabu Search** (TS) [5] manages a memory of solutions or moves recently applied, called the *tabu list*. When a local optimum is reached, the search carries on by selecting a candidate worse than the current solution. To avoid the previous solution to be chosen again, and so to avoid cycles, TS discards the neighboring candidates that have been previously applied.

- **Variable Neighborhood Search**. The basic idea of the Variable Neighborhood Search (VNS) [6] is to successively explore a set of pre-defined neighborhoods to provide a better solution. It uses the descent method to get the local minimum. Then, it explores either at random or systematically the set of neighborhoods. At each step, an initial solution is shaked from the current neighborhood. The current solution is replaced by a new one if and only if a better solution has been found. The exploration is thus re-started from that solution in the first neighborhood. If no better solution is found the algorithm moves to the next neighborhood, randomly generates a new solution and attempts to improve it.

Population based metaheuristics:

- **Evolutionary Algorithms** (broadly called EAs) are stochastic search techniques that have been successfully applied in many real and complex applications (epistatic, multimodal, multi-objective and highly constrained problems). Their success in solving difficult optimization tasks has promoted the research in the field known as *evolutionary computing* (EC) [7]. An EA is an iterative technique that applies stochastic operators on a pool of individuals (the population). Every individual in the population is the encoded version of a tentative solution. Initially, this population is generated randomly. An evaluation function associates a fitness value to every individual indicating its suitability to the problem. There exist several well-accepted subclasses of EAs depending on representation of the individuals or how makes each step of the algorithm. The main subclasses of EAs are the genetic algorithm (GA), evolutionary programming (EP), the evolution strategy (ES), and some others not shown here.

- **Estimated Distribution Algorithms** (EDAs) are a recent type of optimization and learning techniques based in the concept of using a population of tentative solutions to improve the best-so-far optimum for a problem [8]. The key step in this algorithm is to estimate $p^*(x,t)$ and to generate new points according to this distribution. This represents a clear difference with respect to other evolutionary algorithms that use recombination and/or mutation operators to compute a new population of tentative solutions.

- **Scatter Search** (SS) [9] is a population-based metaheuristic that combines solutions selected from a reference set to build others. The method starts by generating an initial population of disperse and good solutions. The reference set is then constructed by selecting good representative solutions from the population. The selected solutions are combined to provide starting solutions to an improvement procedure. According to the result of such procedure the reference set and even the population of solutions can be updated. The process is iterated until a stopping criterion is satisfied. The SS approach involves different procedures allowing to generate the initial population, to build and update the reference set, to combine the solutions of such set, to improve the constructed solutions, etc.

Both approaches -trajectory-based and population-based- can be also combined to yield more powerful optimization techniques. This is particularly the case of **memetic algorithms** (MA) [10], that blend ideas of different metaheuristics within the framework of population-based techniques. This can be done in a variety of ways, but most common approaches rely on the embedding of a trajectory-based technique within an EA-like algorithm, see [11, 12]. It is also worth mentioning those metaheuristics included in the so-called *swarm intelligence* paradigm, such as for example **ant colony optimization** (ACO), and **particle swarm optimization** (PSO). These techniques regard optimization as an emergent phenomenon from the interaction of simple search agents.

1.2.2 Bioinformatic tasks

Now, in this subsection we describe the main bioinformatic tasks, giving, at the of the section, a table with the metaheuristic approaches applied to solve them. Based on the availability of the date and goals, we can classify the problems of bioinformatics as follows:

- Alignment and comparison of Genome and Proteome Sequences: From the biological point of view, sequence comparison is motivated by the fact that all living organism are related by evolution. That implies that the genes of species that are closer to each other should exhibit similarities at DNA level. In biology, the sequences to be compared are either nucleotides (DNA, RNA), or amino acids (proteins). In the case of nucleotides, one usually aligns identical nucleotide symbols. When dealing with amino acids the alignment of two amino acids occurs if they are identical or if one can be derived from the other by substitutions that are likely to occur in nature. The comparison of sequences comprise pairwise and simultaneous multiple sequence comparisons (and alignments). Therefore, algorithms for these problems should allow the deletion, insertion, and replacement of symbols (nucleotides or amino acids) and they should be capable of comparing large number of long sequences. An interesting problem related to both sequence alignment and microarray production (see below) will be described in Section 1.4.

- DNA Fragment Assembly: The fragment assembly problem consists in the building of the DNA sequence from several hundreds (or even, thousands) of fragments obtained by biologists in the laboratory. This is an important task in any genome project since the rest of the phases depend on the accuracy of the results of this stage. This problem will be detailed in Section 1.3.

- Gene Finding and Identification: It is frequently the case in bioinformatics that one wishes to delimit parts of sequences that have a biological meaning. Typical examples are determining the locations of promoters, exons, and introns in RNA. In particular automatic identification of the genes from the large DNA sequences is an important problem. Recognition of regulatory regions in DNA fragments has become particularly popular because of the increasing number of completely sequenced genomes and mass application of DNA chips.

- Gene Expression Profiling: This is the process for determining when and where particular genes are expressed. Furthermore, the expression of one gene is often regulated by the expression of another gene. A detailed analysis of all this information will provide and understanding about the inter-networking of different genes and their functional roles. Microarray technology is used for that purpose.

  Microarray technology allows expression levels of thousand of genes to be measured at the same time. This allows the simultaneous study of tens of thousand of different DNA nucleotide sequences on a single microscopic glass slide. Many important biological results can be obtained by correctly selecting, assembling, analyzing, and interpreting microarray data. Clustering is the most common task, and allows to identify groups of genes that share similar expressions and maybe similar functions.

- Structure Prediction: Determining the structure of protein is very important since there exists a strong relation between the structure and the function. This is one of the most challenging tasks in bioinformatics. There are three main levels of protein structure:

  1. The primary structure is its linear sequence of amino acids.
  2. The secondary structure is the folding of the primary structure via hydrogen bonds.
  3. The tertiary structure refers to the 3-D structure of the protein and it is generated by packing the secondary structural elements. Generally the protein function depends on its tertiary structure.

Table 1.1 Main bionformatic tasks and some representative metaheuristics applied to them.

Bioinformatic Tasks	**Metaheuristics**
Sequence comparison and alignment	*EA* [13, 14] *MA* [15] *ACS* [16] *PSO* [17]
DNA fragment assembly	*EA* [18, 19] *SA* [20] *ACS* [21]
Gene Finding and Identification	*EA* [22, 23]
Gene Expression Profiling	*EA* [24] *MA* [25, 26] *PSO* [27]
Structure Prediction	*EA* [28, 29] *MA* [30] *SA* [31] *EDA* [32]
Phylogenetic Trees	*EA* [33, 34] *SS* [35] *MA* [36]

4. The quaternary structure describe the formation of protein complexes composed of more than one chain of amino acids.

Also, the protein docking problem is related to the structure of the protein. This problem is to determine how interact with other proteins and plays a key role in understanding the protein function.

- Phylogenetic Analysis: All species undergo a slow transformation process called evolution. Phylogenetic trees are labelled binary trees where leaves represent current species and inner nodes represent hypothesized ancestors. Phylogenetic analysis is used to study evolutionary relationships.

1.3 THE DNA FRAGMENT ASSEMBLY PROBLEM

In this section we study the behavior of a several metaheuristics for the DNA fragment assembly problem. The DNA fragment assembly is a problem solved in the early phases of the genome project and thus very important, since the other steps depend on its accuracy. This is an NP-hard combinatorial optimization problem which is growing in importance and complexity as more research centers become involved on sequencing new genomes.

In the next subsection, we present background information about the DNA fragment assembly problem. Later, the details of our approaches are presented and how to design and implement these methods for the DNA fragment assembly problem. We finish this section analyzing the results of our experiments.

1.3.1 Description of the problem

In order to determine the function of specific genes, scientists have learned to read the sequence of nucleotides comprising a DNA sequence in a process called DNA sequencing. To do that, multiple exact copies of the original DNA sequence are made. Each copy is then cut into short fragments at random positions. These are the first three steps depicted in Fig. 1.1 and they take place in the laboratory. After the fragment set is obtained, a traditional assemble approach is followed in this order: overlap, layout, and then consensus. To ensure that enough fragments overlap, the reading of fragments continues until a coverage is satisfied. These steps are the last three ones in Fig. 1.1. In what follows, we give a brief description of each of the three phases, namely overlap, layout, and consensus.

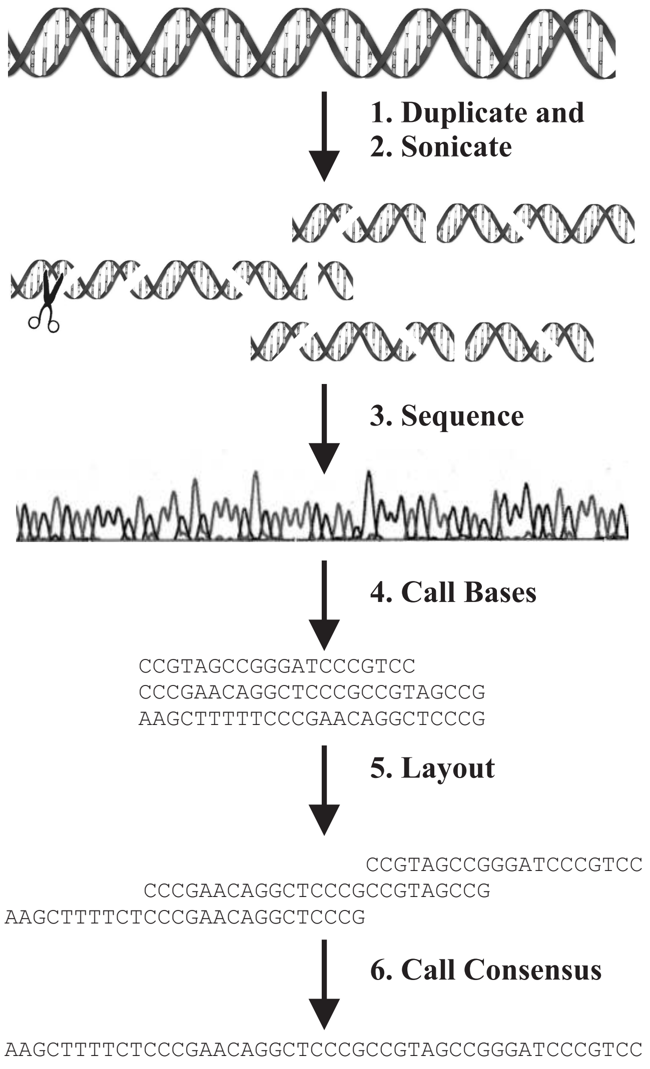
1. Duplicate and
2. Sonicate
3. Sequence
4. Call Bases
CCGTAGCCGGGATCCCGTCC
CCCGAACAGGCTCCCGCCGTAGCCG
AAGCTTTTTCCCGAACAGGCTCCCG
5. Layout
CCGTAGCCGGGATCCCGTCC
CCCGAACAGGCTCCCGCCGTAGCCG
AAGCTTTTCTCCCGAACAGGCTCCCG
6. Call Consensus
AAGCTTTTCTCCCGAACAGGCTCCCGCCGTAGCCGGGATCCCGTCC

Fig. 1.1 Graphical representation of DNA sequencing and assembly

Overlap Phase - Find the overlapping fragments. This phase consists in finding the best or longest match between the suffix of one sequence and the prefix of another. In this step, we compare all possible pairs of fragments to determine their similarity. Usually, a dynamic programming algorithm applied to semiglobal alignment is used in this step. The intuition behind finding the pairwise overlap is that fragments with a high overlap are very likely next to each other in the target sequence.

Layout Phase - Find the order of fragments based on the computed similarity score. This is the most difficult step because it is hard to tell the true overlap due to the following challenges:

1. Unknown orientation: After the original sequence is cut into many fragments, the orientation is lost. One does not know which strand should be selected. If one fragment does not have any overlap with another, it is still possible that its reverse complement might have such an overlap.

2. Base call errors: There are three types of base call errors: substitution, insertion, and deletion. They occur due to experimental errors in the electrophoresis procedure (the method used in the laboratories to read the ADN sequences). Errors affect the detection of fragment overlaps. Hence, the consensus determination requires multiple alignments in highly coverage regions.

3. Incomplete coverage: It happens when the algorithm is not able to assemble a given set of fragments into a single contig. A contig is a sequence in which the overlap between adjacent fragments is greater or equal to a predefined threshold (cutoff parameter).

4. Repeated regions: “Repeats” are sequences that appear two or more times in the target DNA. Repeated regions have caused problems in many genome-sequencing projects, and none of the current assembly programs can handle them perfectly.

5. Chimeras and contamination: Chimeras arise when two fragments that are not adjacent or overlapping on the target molecule join together into one fragment. Contamination occurs due to the incomplete purification of the fragment from the vector DNA.

After the order is determined, the progressive alignment algorithm is applied to combine all the pairwise alignments obtained in the overlap phase.

Consensus Phase - Derive the DNA sequence from the layout. The most common technique used in this phase is to apply the majority rule in building the consensus.

To measure the quality of a consensus, we can look at the distribution of the coverage. Coverage at a base position is defined as the number of fragments at that position. It is a measure of the redundancy of the fragment data, and it denotes the number of fragments, on average, in which a given nucleotide in the target DNA is expected to appear. It is computed as the number of bases read from fragments over the length of the target DNA [37].

$$Coverage = \frac{\sum_{i=1}^{n} length\ of\ the\ fragment\ i}{target\ sequence\ length} \tag{1.1}$$

where n is the number of fragments. The higher the coverage, the fewer number of the gaps, and the better the result.

1.3.2 DNA Fragment Assembly Using Metaheuristics

Let us give some details about the most important issues of our implementation and how we have used the some metaheuristics to solve the DNA fragment assembly problem. First, we address the common details such as the solution representation or the fitness function, and then, we describe the specific features of each algorithm. The methods used are a genetic algorithm (GA) [38], a CHC method [39], a scatter search (SS) [9], and a simulated annealing (SA) [4] (for more detail about these algorithms, we refer the reader to [3].

Common Issues

- Solution Representation: We use the permutation representation with integer number encoding. A permutation of integers represents a sequence of fragment numbers, where successive fragments overlap. The solution in this representation requires a list of fragments assigned with a unique integer ID. For example, 8 fragments will need eight identifiers: 0, 1, 2, 3, 4, 5, 6, 7. The permutation representation requires special operators to make sure that we always get legal (feasible) solutions. In order to maintain a legal solution, the two conditions that must be satisfied are (1) all fragments must be presented in the ordering, and (2) no duplicate fragments are allowed in the ordering.

- Fitness Function: A fitness function is used to evaluate how good a particular solution is. In the DNA fragment assembly problem, the fitness function measures the multiple sequences alignment quality and finds the best scoring alignment. Our fitness function [18] sums the overlap score ($w(f, f1)$) for adjacent fragments ($f[i]$ and $f[i+1]$) in a given solution. When this fitness function is used, the objective is to maximize such a score. It means that the best individual will have the highest score, since the order proposed by that solution has strong overlap between adjacent fragments.

$$F1(l) = \sum_{i=0}^{n-2} w(f[i]f[i+1]) \tag{1.2}$$

- Program Termination: The program can be terminated in one of two ways. We can specify the maximum number of evaluations to stop the algorithm or we can also stop the algorithm when the solution is no longer improving.

GA Details

- Population Size: We use a fixed size population to initialize random solutions.

- Recombination Operator: Two or more parents are recombined to produce one or more offspring. The purpose of this operator is to allow partial solutions to evolve in different individuals and then combine them to produce a better solution. It is implemented by running through the population and for each individual, deciding whether it should be selected for crossover using a parameter called *crossover rate* (P_c). For our experimental runs, we use the order-based crossover (OX). This operator first copies the fragment ID between two random positions in Parent1 into the offspring's corresponding positions. We then copy the rest of the fragments from Parent2 into the offspring in the relative order presented in Parent2. If the fragment ID is already present in the offspring, then we skip that fragment. The method preserves the feasibility of every tentative solution in the population.

- Mutation Operator: This operator is used for the modification of single individuals. The reason we need a mutation operator is for the purpose of maintaining diversity in the population. Mutation is implemented by running through the whole population and for each individual, deciding whether to select it for mutation or not, based on a parameter called *mutation rate* (P_m). For our experimental runs, we use the swap mutation operator and invert segment mutation operator. The first operator randomly selects two positions from a permutation and then swaps the two fragment positions. The second one also selects two positions from a permutation and then inverts the order of the fragments in partial permutation defined by the two random positions (i.e., we swap two edges in the equivalent graph). Since these operators do not introduce any duplicate number in the permutation, the solution they produce is always feasible.

- Selection operator: The purpose of the selection is to weed out the bad solutions. It requires a population as a parameter, processes the population using the fitness function, and returns a new population. The level of the selection pressure is very important. If the pressure is too low, convergence becomes very slow. If the pressure is too high, convergence will be premature to a local optimum.

  In this work, we use ranking selection mechanism [40], in which the GA first sorts the individuals based on the fitness and then selects the individuals with the best fitness score until the specified population size is reached. Preliminary results favored this method out of a set of other selection techniques analyzed. Note that the population size will grow whenever a new offspring is produced by crossover or mutation operators.

CHC Details

- Incest Prevention: CHC method has a mechanism of *incest prevention* to avoid recombination of similar solutions. Typically, it is used the hamming distance as measure of similarity but this one is unsuitable for permutations. In the experiments, we consider that the distance between two solutions is the total number of edges minus the number of common edges.
- Recombination: The crossover that we use in our CHC, creates a single child by preserving the edges that parents have in common and the randomly assigning the remaining edges in order to generate a legal permutation.
- Population restart: Whenever the population converges, the population is partially randomized for a restart by using the best individual found so far as a template and creating new individuals by repeatedly swapping edges until a specific fraction of edges differ from those template.

SS Details

- Initial population creation: There exist several ways to get an initial population of good and disperse solutions. In our experiments, the solutions for the population was randomly generated to achieve a certain level of diversity. Then, we apply the *Improvement method* (that it will be explained in next section) to these solutions in order to get better solutions.
- Subsets generation and Recombination operator: It generates all 2-elements subsets and then it applies the recombination operator to them. For our experimental runs, we use the order-based crossover that it was explained in previous section.
- Improvement method: We apply a hillclimber procedure to improve the solutions. The hillclimber is a variant of Lin's two-opt [41]. Two position are randomly selected, and then it inverts the subpermutation by swapping the two edges. Whenever an improvement is found, the solution is updated, and the hillclimber continues until it achieves a predetermined number of swap operations.

SA Details

- Cooling scheme: The cooling schedule controls the values of the temperature parameter. It specifies the initial value and how the temperature is updated at each stage of the algorithm.

  $$T_k = \alpha * T_{k-1} \tag{1.3}$$

  In this case, we use a decreasing function (Eq. 1.3) controlled by the α factor where $\alpha \in (0, 1)$.

- Markov chain length: The number of the iterations between two consecutive changes of the temperature is given by the parameter $Markov\_Chain\_length$, whose name alludes the fact that the sequence of accepted solutions is a Markov chain (a sequence of states in which each state only depends on the previous one).
- Move operator: This operator generates a new neighbor from current solution. For our experimental runs, we use the edge swap operator. This operator randomly selects two positions from a permutation and then invert the order of the fragment between these two fragment positions.

1.3.3 Experimental Results

A target sequence with accession number BX842596 (GI 38524243) was used in this work. It was obtained from the NCBI web site[1]. It is the sequence of a Neurospora crassa (common bread mold) BAC, and is 77,292 base pairs long. To test and analyze the performance of our algorithm, we generated two problem instances with GenFrag [42]. GenFrag takes a known DNA sequence and uses it as a parent strand from which to randomly generate fragments according to the criteria (mean fragment length and coverage of parent sequence) supplied by the user. The first problem instance, 842596_4, contains 442 fragments with average fragment length of 708 bps and coverage 4. The second problem instance, 842596_7, contains 773 fragments with average fragment length of 703 bps and coverage 7. We evaluated the results in terms of the number of contigs assembled.

We use a GA, a CHC, a SS, and a SA to solve this problem. To allow a fair comparison among the results of these heuristics, we have configured them to perform a similar computational effort (the maximum number of evaluations for any algorithm is 512000). Since the results of these algorithms vary depending on the different parameter settings, we previously performed a complete analysis to study how the parameters affect the performance of the algorithms. A summary of the conditions for our experimentation is found in Table 1.2. We have performed statistical analyses to ensure the significance of the results and to confirm that our conclusions are valid (all the results are statistically different).

Table 1.3 shows all the results and performance with all data instances and algorithms described in this work. The table shows the fitness of the best solution obtained (b), the average fitness found (f), average number of evaluations (e), and average time in seconds (t). We do not show the standard deviation because the fluctuations in the accuracy of different runs are rather small, claiming that the algorithms are very robust (as proved by the ANOVA results). The best results are boldfaced.

Let us discuss some of the results found in this table. First, for the two instances, it is clear that the SA outperforms the rest of the algorithms from every point of view. In fact, SA obtains better fitness values than the previous best known solutions [20].

[1] `http://www.ncbi.nlm.nih.gov`

Table 1.2 Parameters when heading and optimum solution of the problem.

Common Parameters	
Independent runs	30
Cutoff	30
Max Evaluations	512000
Genetic Algorithms	
Popsize	1024
Crossover	OX (0.8)
Mutation	Edge Swap (0.2)
Selection	Ranking
CHC	
Popsize	50
Crossover	specific (1.0)
Restart	Edge Swap (30%)
Scatter Search	
Initial Popsize	15
Reference set	8 (5 + 3)
Subset generation	All 2-elements subsets (28)
Crossover	OX (1.0)
Improvement	Edge Swap (100 iterations)
Simulated Annealing	
Move operator	Edge Swap
Markov chain length	$\frac{total\ number\ evaluations}{100}$
Coolin scheme	Proportional ($\alpha = 0.99$)

Also, its execution time is the lowest one. The reason of that is that the SA operates on a single solution, while the rest of the methods are population-based and in addition they execute time-consuming operators (specially the crossover operation).

The CHC is the worst algorithm in both, execution time and solution quality. Its larger runtime is due to the additional computations needed to detect the converge of the population or to detect incest mating. CHC is not able of solving the DNA fragment assembly problem adequately and maybe it needs a local search (as proposed by [39]) to reduce the search space.

Table 1.3 Results on the Two Instances.

	38524243_4				**38524243_7**			
Algorithm	b	f	e	t	b	f	e	t
Genetic Algorithm	92772	88996	508471	32.62	108297	104330	499421	85.77
CHC	61423	54973	487698	65.33	86239	81943	490815	162.29
Scatter Search	94127	90341	**51142**	27.83	262317	254842	**52916**	66.21
Simulated Annealing	**225744**	**223994**	504850	**7.92**	**416838**	**411818**	501731	**12.52**

The SS obtains better solutions than the GA, and it also achieves these solutions in a smaller runtime. This means that a structured application of operator and explicit reference search points are good idea for this problem.

The computational effort to solve this problem (only considering the number of evaluations) is similar for all the heuristics with the exception of the SS, because its most time-consuming operation is the improvement method that does not perform complete evaluations of the solutions. This result indicates that all the algorithms examine a similar number of points in the search space, and the difference in the solution quality is due to how they explore the search space. For this problem, trajectory based methods such as the simulated annealing are more effective than population based ones. Thus, the resulting ranking of algorithms from best to worse is SA, SS, GA, and finally CHC.

Table 1.4 Final Best Contigs.

Algorithms	**38524243_4**	**38524243_7**
GA	6	4
CHC	7	5
SS	6	4
SA	**4**	**2**

Finally, Table 1.4 shows the final number of contigs computed in every case. A contig is a sequence in which the overlap between adjacent fragments is greater than a threshold (cutoff parameter). Hence, the optimum solution has a single contig. This value is used as a high-level criterion to judge the whole quality of the results since, as we said before, it is difficult to capture the dynamics of the problem into a mathematical function. These values are computed by applying a final step of refinement with a greedy heuristic popular in this application [19]. We have found that in some (extreme) cases it is possible that a solution with a better fitness than other one generates a larger number of contigs (worse solution). This is the reason for still needing more research to get a more accurate mapping from fitness to contig number. The values of this table however confirm again that the SA method outperform the rest clearly, the CHC obtains the worst results, and the SS and GA obtain similar number of contigs.

1.4 THE SHORTEST COMMON SUPERSEQUENCE PROBLEM

The Shortest Common Supersequence Problem (SCSP) is a classical problem from the realm of string analysis. Roughly speaking, the SCS problem amounts to finding a minimal-length sequence $S \in \Sigma^*$ of symbols from a certain alphabet Σ, such that every sequence in a certain set $L \in 2^{\Sigma^*}$ can be generated from S by removing some symbols of the latter. The resulting combinatorial problem is enormously interesting from the point of view of bioinformatics [43], and bears close relationship with sequence alignment and microarray production among other tasks.

Unfortunately, the SCS problem has been shown to be hard under various formulation and restrictions, resulting not just in a NP-hard problem, but also in a non-FPT problem [44], so practical resolution is likely unaffordable by conventional exact techniques. Therefore, several heuristics and metaheuristics have been defined to tackle it. Before detailing these, let us firstly describe more formally the SCSP.

1.4.1 Description of the Problem

We write $|s|$ for the length of sequence s ($|s_1 s_2 \dots s_n| = n$), and $|\Sigma|$ for the cardinality of set Σ. Let ϵ be the empty sequence ($|\epsilon| = 0$). We use $s \trianglerighteq \alpha$ for the total number of occurrences of symbol α in sequence s:

$$s_1 s_2 \dots s_n \trianglerighteq \alpha = \sum_{1 \le i \le n, s_i = \alpha} 1.$$

We write αs for the sequence obtained by appending the symbol α in front of sequence s. Deleting symbol α from the front of sequence s is denoted by $s|_\alpha$. We also use the $|$ symbol to delete a symbol from the front of a set of strings: $\{s_1, \cdots, s_m\}|_\alpha = \{s_1|_\alpha, \cdots, s_m|_\alpha\}$.

Sequence s is said to be a supersequence of r (denoted as $s \succ r$) if $r = \epsilon$, or if $s|_{s_1}$ is a supersequence of $r|_{s_1}$. Plainly, $s \succ r$ implies that r can be embedded in s, meaning that all symbols in r are present in s in the same exact order, although not necessarily consecutive. 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 sequences $\{s_1, \cdots, s_m\}$, $s_i \in \Sigma^*$. The problem consists of finding a sequence s of minimal length that is a supersequence of each sequence in L ($s \succ s_i, \forall s_i \in L$ and $|s|$ is minimal).

A particularly interesting situation for bioinformatics arises when the sequences represent molecular data, i.e., sequences of nucleotides or amino acids.

1.4.2 Heuristics and Metaheuristics for the SCSP

One of the most simple and effective algorithms for the SCSP 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 sequences in L, and removing these symbols from the corresponding strings. Ties can be randomly broken, and the process is repeated until all sequences in L are empty. A drawback of this algorithm is its myopic functioning, that 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 [45]. 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 in whose front is located. Branke *et al.* [45] propose to use precisely this string length as weight (i.e., the weight of a symbol is the sum of the length of all strings at whose front it appears). 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 [46, 45].

The limitations of these simple heuristics have been dealt via the use of more sophisticated techniques. One of the first metaheuristic approaches to the SCSP is due to Branke *et al.* [45]. They consider several evolutionary algorithms approaches, and in particular a GA that uses WMM as decoding mechanism. More precisely, the GA evolves weights (or meta-weights actually) that are used to further refine the weights computed by the WMM heuristic. This latter algorithm is then used to compute tentative supersequences on the basis of these modified weights (this procedure is similar to the EA used in [47] 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). A related approach is presented in [48] based on ant colony optimization (ACO). Pheromone values take the role of weights, and ants take probabilistic decisions based on these weights. An interesting feature of this ACO approach is the fact that pheromone update is not done on an individual basis (i.e., just on the symbols being picked each time), but globally (i.e., whenever a symbol is picked, the pheromone of previous symbols that allowed that symbol being picked is also increased).

More recent approaches to the SCSP are based on EAs [46]. These EAs range from simple direct approaches based on penalty functions, to more complex approaches based on repairing mechanisms or indirect encodings:

- Simple EA: solutions are represented as a sequence of symbols corresponding to a tentative supersequence. Fitness is computed as the length of the supersequence, plus a penalty term in case a solution does not contain a valid supersequence. This penalty term is precisely the length of the solution provided by the MM heuristic for the remaining sequences, i.e.,

$$F\left(s,L\right)=\begin{cases}0 & \text{if } \forall i: s_i=\epsilon\\ 1+F(s',L|_\alpha) & \text{if } \exists i: s_i\neq\epsilon \text{ and } s=\alpha s'\\ |\mathrm{MM}(L)| & \text{if } \exists i: s_i\neq\epsilon \text{ and } s=\epsilon\end{cases} \tag{1.4}$$

  Standard recombination and mutation operators can be used in this EA.

- EA + repairing: based on the simple EA, a repair function ρ is used to complete the supersequence with the symbols indicated by the MM heuristic, i.e.,

$$\rho\left(s,L\right)=\begin{cases}s & \text{if } \forall i: s_i=\epsilon\\ \rho(s',L) & \text{if } \exists i: s_i\neq\epsilon \text{ and } \nexists i: s_i=\alpha s'_i \text{ and } s=\alpha s'\\ \alpha\rho(s',L|_\alpha) & \text{if } \exists i: s_i=\alpha s'_i \text{ and } s=\alpha s'\\ \mathrm{MM}(L) & \text{if } \exists i: s_i\neq\epsilon \text{ and } s=\epsilon\end{cases} \tag{1.5}$$

Notice that this function not only turns infeasible solutions to feasible ones, but it also improves feasible solutions by removing unproductive steps.

- indirect EAs: in the line of the GRASP-like decoding mechanism defined in [49], the EA evolves sequences of integers that denote the rank of the symbol that has to be picked at each step, on the basis of the ranking provided by a simple heuristic such as MM or WMM.

The experimental results [46] indicate that the EA with repairing is better than the other two approaches for small alphabet sizes, e.g., in the case of nucleotide sequences. for larger alphabets, the EA with indirect encoding is slightly better. Further improvements are obtained from the inclusion of a local search technique within the EA, thus resulting in a MA [15]. Local search is done by removing symbols from the supersequence, and retaining the deletion if the resulting solution is valid. This local search procedure is costly, so it has to be used with caution. To be precise, it is shown that partial lamarckism rates -around 1% or 5%- provide the best tradeoff between computational cost and quality of results.

A completely different metaheuristic - beam search (BS)- has been also proposed for the SCSP in [50]. This technique approaches the incremental construction of k (a parameter) solutions via a pseudo-population-based greedy mechanism inspired in branch and bound, and the breadth-first traversal of search trees. This is a fully deterministic approach (save for tie breaking) that is shown to provide moderately good solutions to the problem. Several improvements to this basic technique have been proposed. On one hand, the hybridization of BS and MAs has been also attempted in [50]. On the other hand, a probabilistic version of BS (PBS) has been described in [51]. Both approaches have been shown to be superior to the basic technique, and are very competitive in general. PBS is remarkable for its success in small sequences, but starts to suffer scalability problems when the length of sequences increases.

1.5 EXPERIMENTAL RESULTS

A experimental comparison has been made between the most competitive algorithms described before, namely the MA-BS hybrid [50] and both the simple heuristics MM and WMM. The instances considered for the experimentation 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]:

- *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 amino acid long.

[2]http://gel.ym.edu.tw/sars/genomes.html
[3]http://www.expasy.org/sprot/

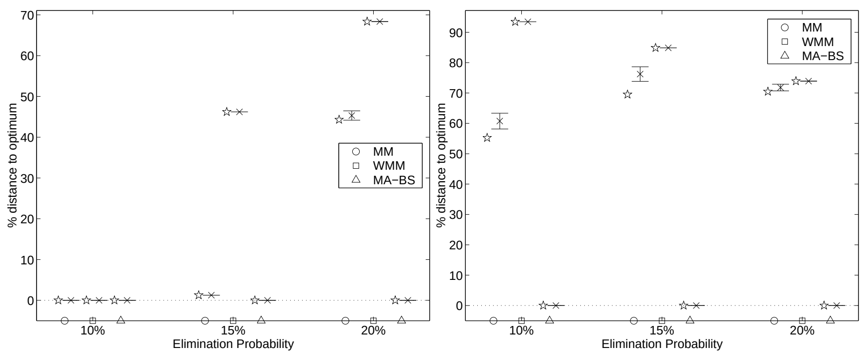

Fig. 1.2 Results of MM, WMM, and MA-BS on the SARS158 and SARS1269 instances. Each group of bars indicate the results of the corresponding algorithm (MM, WMM and MA-BS) for the three different values of p (i.e., $p = \{5\%, 10\%, 20\%\}$). The star indicates the best solution found by the corresponding algorithm, the cross marks the mean, and the bars indicate the standard deviation.

- *p53*: this protein is involved in the cell cycle, and acts as tumor suppressor in many tumor types; the sequence is 393 amino acid long.

- *Estrogen*: involved in the regulation of eukaryotic gene expression, this protein affects cellular proliferation and differentiation; the sequence is 595 amino acid long.

In all cases, problem instances are constructed 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 $p = \{5\%, 10\%, 20\%\}$, thus representing increasingly hard optimization problems.

All algorithms are run for 600 seconds on a P4 2.4GHz 512MB computer. The particular parameters of the MA component are *popsize*$= 100$, $p_X = .9$, $p_M = 1/L$, $p_{LS} = .01$, uniform crossover, tournament selection, and steady-state replacement. As to the BS component, it considers $k = 10,000$. The results (averaged for 20 runs) are shown in Fig. 1.2-1.3.

As it can be seen, the conventional greedy heuristics cannot compete with the hybrid approach. Notice that the results of the latter ones are very close to the putative optimal (the original sequence to be reconstructed). Actually, it can be seen that the hybrid MA-BS algorithm manages to find this original solution in all runs. As reported in [51], the PBS algorithm is also capable of performing satisfactorily for most of these problem instances, but is affected by the increased dimensionality of the problem in the SARS 1269 instance, for high values of p. MA-BS is somewhat more robust in this sense.

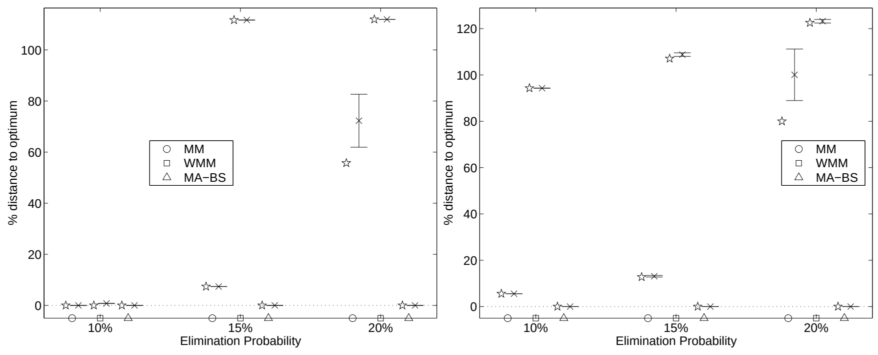

Fig. 1.3 Results of MM, WMM, and MA-BS on the P53 and Estrogen instances; all techniques manage to find the optimal solution in most runs for the Oxytocin instance. The exceptions are MM and WMM for the 20% case.

1.6 CONCLUSIONS

The increasing of biological data and the needs to analyze and interpret them have opened several important research lines in computation science since the application of computational approaches allows to facilitate the understanding of various biological process. In this chapter we have shown how these problems can be solved with metaheuristics. In particular, we have tackled two difficult problems: the DNA fragment assembly problem and the shortest common supersequence problem.

Both the DNA FAP and the SCSP are very complex problems in computational biology. We tackled the first problem with four sequential heuristics: three population based ones (genetic algorithm, scatter search, and CHC), and a trajectory based method (simulated annealing). We have observed that this last outperforms the rest of the methods (what implies more research in trajectory based methods in the future). It obtains a much better solutions than the others and it runs faster. This local search algorithm outperformed here the best known fitness values in the literature. As to the second one, we have shown that the best approach described in the literature is the hybrid of memetic algorithms and beam search defined by Gallardo *et al.* [50]. This hybrid method vastly outperforms conventional greedy heuristics on both DNA sequences and protein sequences.

Acknowledgments

The authors are partially supported by the Ministry of Science and Technology and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). Second and third authors are also supported under contract TIN-2007-67134.

References

1. J. Cohen. Bioinformatics – an introduction to computer scientists. *ACM Computing Surveys*, 36:122–158, 2004.
2. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
3. Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer, January 2003.
4. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
5. F. Glover. Tabu Search, part I. *ORSA, J. of Computing*, 1:190–206, 1989.
6. N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers Oper. Res*, 24:1097–1100, 1997.
7. T. Bäck, D.B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, 1997.
8. H. Mühlenbein, T. Mahnig, and A. Ochoa. Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, 5(2):215–247, 1999.
9. F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39(3):653–684, 2000.
10. P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston MA, 2003.
11. P. Moscato, C. Cotta, and A. Mendes. Memetic algorithms. In G.C. Onwubolu and B.V. Babu, editors, *New Optimization Techniques in Engineering*, pages 53–85. Springer-Verlag, Berlin Heidelberg, 2004.
12. P. Moscato and C. Cotta. Memetic algorithms. In T. González, editor, *Handbook of Approximation Algorithms and Metaheuristics*, chapter 27. Taylor & Francis, 2007.

13. C. Notredame and D.G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24:1515–1524, 1996.

14. C. Zhang and A. K. Wong. A genetic algorithm for multiple molecular sequence alignment. *Comput Appl Biosci.*, 13(6):565–581, 1997.

15. C. Cotta. Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In J. Mira and J.R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, volume 3562 of *Lecture Notes in Computer Science*, pages 84–91, Berlin Heidelberg, 2005. Springer-Verlag.

16. O. Karpenko, J. Shi, and Y. Dai. Prediction of mhc class ii binders using the ant colony search strategy. *Artificial Intelligence in Medicine*, 35(1-2):147–156, 2005.

17. T.K. Rasmussen and T. Krink. Improved Hidden Markov Model training for multiple sequence alignment by a particle swarm optimization-evolutionary algorithm hybrid. *Biosystems*, 72(1-2):5–17, 2003.

18. R. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and DNA fragment assembly. *Machine Learning*, 21:11–33, 1995.

19. L. Li and S. Khuri. A Comparison of DNA Fragment Assembly Algorithms. In *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335, 2004.

20. E. Alba, G. Luque, and S. Khuri. Assembling DNA Fragments with Parallel Algorithms. In B. McKay, editor, *CEC-2005*, pages 57–65, Edinburgh, UK, 2005.

21. P. Meksangsouy and N. Chaiyaratana. DNA fragment assembly using an ant colony system algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC ’03*, volume 3, pages 1756– 1763. IEEE Press, 2003.

22. A. Kel, A. Ptitsyn, V. Babenko, S. Meier-Ewert, and H. Lehrach. A genetic algorithm for designing gene family-specific oligonucleotide sets used for hybridization: The g protein-coupled receptor protein superfamily. *Bioinformatics*, 14:259–270, 1998.

23. V.G. Levitsky and A.V. Katokhin. Recognition of Eukaryotic Promoters Using a Genetic Algorithm Based on Iterative Discriminant Analysis. *In Silico Biology*, 3(1):81–87, 2003.

24. K. Deb, S. Agarwal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *Proceedings of the Parallel Problem Solving from Nature VI Conferences*, pages 849–858, 2000.

25. C. Cotta, A. Mendes, V. Garcia, P. Franca, and P. Moscato. Applying memetic algorithms to the analysis of microarray data. In G. Raidl et al., editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 22–32, Berlin, 2003. Springer-Verlag.

26. A. Mendes, C. Cotta, V. Garcia, P. França, and P. Moscato. Gene ordering in microarray data using parallel memetic algorithms. In T. Skie and C.-S. Yang, editors, *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, pages 604–611, Oslo, Norway, 2005. IEEE Press.

27. X. Xiao, E.R. Dow, R. Eberhart, Z.B. Miled, and R.J. Oppelt. Gene clustering using self-organizing maps and particle swarm optimization. In *Sixth IEEE International Workshop on High Performance Computational Biology*, Nice, France, April 2003.

28. C. Notredame, L. Holm, and D.G. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

29. G. Fogel and D. Corne. *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann, San Francisco, 2002.

30. C. Cotta. Protein structure prediction using evolutionary algorithms hybridized with backtracking. In J. Mira and J.R. Álvarez, editors, *Artificial Neural Nets Problem Solving Methods*, volume 2687 of *Lecture Notes in Computer Science*, pages 321–328, Berlin Heidelberg, 2003. Springer-Verlag.

31. J.J. Gray, S. Moughon, C. Wang, O. Schueler-Furman, B. Kuhlman, C.A. Rohl, and D. Baker. Protein-protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. *Journal of Molecular Biology*, 331(1):281–299, 2003.

32. R. Santana, P. Larrañaga, and J.A. Lozano. Protein folding in 2-dimensional lattices with estimation of distribution algorithms. In J.M Barreiro, F. Martín-Sánchez, V. Maojo, and F. Sanz, editors, *5th Biological and Medical Data Analysis International Symposium*, volume 3337 of *Lecture Notes in Computer Science*, pages 388–398, Berlin Heidelberg, 2004. Springer-Verlag.

33. P.O. Lewis. A genetic algorithm for maximum likelihood phylogeny inference using nucleotide sequence data. *Mol. Biol. Evol.*, 15:277–283, 1998.

34. C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J.J. Merelo and et al., editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 720–729, Berlin, 2002. Springer-Verlag.

35. C. Cotta. Scatter search with path relinking for phylogenetic inference. *European Journal of Operational Research*, 169(2):520–532, 2006.

36. J. E. Gallardo, C. Cotta, and A. J. Fernández. Reconstructing phylogenies with memetic algorithms and branch and bound. In S. Bandyopadhyay, U. Maulik, and J. T. L. Wang, editors, *Analysis of Biological Data: A Soft Computing Approach*, pages 59–84. World Scientific, 2007.

37. J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*, chapter 4 - Fragment Assembly of DNA, pages 105–139. University of Campinas, Brazil, 1997.

38. J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan, 1975.

39. L.J. Eshelman. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In *Foundations of Genetic Algorithms, 1*, pages 265–283. Morgan Kaufmann, 1991.

40. D. Whitely. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kaufmann, 1989.

41. S. Lin and B.W. Kernighan. An effective heuristic algorithm for tsp. *Operations Research*, 21:498–516, 1973.

42. M.L. Engle and C. Burks. Artificially generated data sets for testing DNA fragment assembly algorithms. *Genomics*, 16, 1993.

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

44. R. Downey and M. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

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

46. C. Cotta. A comparison of evolutionary approaches to the shortest common supersequence problem. In J. Cabestany, A. Prieto, and D.F. Sandoval, editors, *Computational Intelligence and Bioinspired Systems*, volume 3512 of *Lecture Notes in Computer Science*, pages 50–58, Berlin Heidelberg, 2005. Springer-Verlag.

47. C. Cotta and J.M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 3*, pages 251–255, Wien New York, 1998. Springer-Verlag.

48. R. Michel and M. Middendorf. An ant system for the shortest common supersequence problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 51–61. McGraw-Hill, London, 1999.

49. C. Cotta and A. Fernández. A hybrid GRASP - evolutionary algorithm approach to golomb ruler search. In X. Yao et al., editors, *Parallel Problem Solving From Nature VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 481–490, Berlin, 2004. Springer-Verlag.

50. 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*, 37(1):77–83, 2007.

51. C. Blum, C. Cotta, A.J. Fernández, and J.E. Gallardo. A probabilistic beam search approach to the shortest common supersequence problem. In C. Cotta and J. van Hemert, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 36–47, Berlin Heidelberg, 2007. Springer-Verlag.