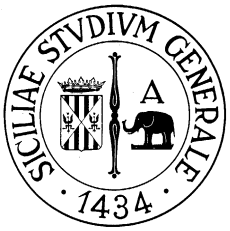


UNIVERSITY OF CATANIA
Department of Mathematics and Computer Science

ELISA PAPPALARDO

# COMBINATORIAL OPTIMIZATION METHODS FOR PROBLEMS IN GENOMICS

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

PhD Course in Computer Science - XXIV Ciclo

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Prof. Domenico Cantone) Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Prof. Domenico Cantone) Director of Graduate Studies

Approved for the University Committee on Graduate Studies.

# Acknowledgements

A PhD thesis is only the final result of years of work and commitment to research, therefore this work would not have been possible without the precious advices and contributions of many people.

First, I would like to thank my Advisors, for their priceless support and encouragement. I am extremely grateful to Professor Cantone, who has followed me since I was a student of Master's Degree. His guidance and support have been fundamental in my path to Doctorate, his "rigorous" approach to research has been a continuous incentive to give the best and to face always challenging new targets. *Grazie.*

I have no words to express my gratitude to Professor Pardalos, I am blessed for having had the chance to work with such a visionary scientist. He welcomed me to his Center for Applied Optimization, at University of Florida; his kindness and his "smiling" helpfulness, along with his "genius", make the lab not only a pleasant and stimulating environment, but also a family of people from every corner of the world. His advices and suggestions will be a continuous source of inspiration for my research. *Ευχαριστό.*

If I am sitting here writing my thesis, a great acknowledgment is due to Giovanni. He has been my first supporter, he encouraged me in every step of the way, and every hill I met, he held my hand and walked with me. I can feel his hand on my shoulder even when an ocean separates us. He is, without any doubt, the *global optimum* in my life.

My family, my parents and my sister Laura were an irreplaceable source of energy and support for me. They patronized all my decisions; they grabbed my hand in their hands for teaching me how to walk, then they let me run on my own feet, and they would give me even "wings" if I just wished to fly. I wish one day I will have such a wonderful family, as mine is for me.

A special thanks to my friends, those speaking the same language I do, and those coming from all around the world. If I had not met such amazing people while I was in Florida, everything would have been much more difficult and sad. Thank you for the precious moments we shared, for the special food you prepared for me, for listening my crying and rejoicing of my happiness. And a special thank to my "Turkish sister", who has made my days full of joy, good food, and priceless friendship. *Teşekkür ederim* Beyza.

# Contents

# List of Tables

# List of Figures

# List of Algorithms

# 1 Introduction

*"What more powerful form of study of mankind could there be than to read our own instruction book?"*

Francis S. Collins,
Director of the Human Genome Project

**From *philosophy* to *genetics*** The word *gene* comes from the Greek term *genesis* (γένεσις), that means *birth, origin.* As a matter of fact, the first speculative theories on the origin of living beings, therefore a "primitive notion of genetics", can be already found in the fourth century B.C., when the Greek philosophers Hippocrates, Socrates, Plato and Aristotle, began to raise fundamental questions about the mechanisms of reproduction and heredity.
Nevertheless, the origin of modern genetics can be dated in the second half of the 19th century, when Gregor Mendel performed a remarkable series of hybridization experiments on pea plants, that led to the two laws of genetics, concerning the basic mechanisms of inheritance in nature. It has been only at the beginning of the past century that genetics has adopted its modern sense: the actual term *genetics* was coined in 1906 by William Bateson, as *"a new and well developed branch of Physiology"*; three years later, the word *gene* was introduced by Wilhelm L. Johannsen for the fundamental physical and functional unit of heredity [Kel02]. With H. J. Muller, a famous geneticist and Nobel laureate of the first half of the 20th century, the word *gene* assumed a more meaningful connotation: it was defined not only as *"the fundamental unit of heredity"*, but as *"the basis of life"* [M$^+$29]. However, these notions were still far from a clear and unique definition of *what a gene was.* To have a definitive answer to the question, we have to wait until

the discovery of the DNA and its structure by Watson and Crick in 1953 [WC53], when a gene was finally identified as a segment of deoxyribonucleic acid, corresponding to a unit of inheritance.

**From *gene* to *genome*** The last few years represent the most flourishing period for biology and related disciplines; in fact, the advances in this field conducted research from pure biological sciences to other new promising areas. Starting from the discovery of DNA and its functions, a new era in biological sciences and medicine has been heralded: from the diagnosis to the detection of genetic predispositions to diseases; from the design of new "custom drugs" based on specific genetic profiles, to the application of gene treatments; from forensic identification to anthropology, DNA has had a radical and widespread impact on many scientific fields. As a consequence, several new research areas have been defined, such as genetic engineering, genomics and proteomics, each of them focusing on specific problems and aspects in genetics. In particular, genomics concerns the study of the genome, that is the entirety of the hereditary information in an organism, including mapping the genes and sequencing the DNA.

**From *biology* to *computational science*** Besides these new branches of biology, the success of the Human Genome Project (HGP, for short) led to a new era also in computational science. HGP was an international 13-years long project, begun formally in 1990, with the goal of identifying all the genes and determining the sequences of the chemical base pairs in human DNA. Due to the great amount of data produced, effective and efficient methodologies are required to perform accurate analysis. In this scenario, computational biology, mathematics, engineering and computer science, bring significant contributions, by suggesting computational methods and *in silico* approaches to handle genomic issues.
In general, computational science can be defined as the discipline characterized by the use of computers to analyze complex physical systems and provide insight of their behavior [Ste93]. In particular, computational science uses advanced computing and data analysis to conduct *in silico* experiments, which are too expensive or impossible to lead in the real world. To approach real-world problems from a computational perspective, a broad range of approaches and mathematical methods have been proposed, ranging from computational mathematics to numerical algorithm.

In the last years, the rapid growth in computer power enabled an increasing and fast development in this new field of science: high-performance calculators, a continuous increment of processor speed, massive parallelization, the emergence of languages and software tools for complex computers, allowed to study and solve highly complex real-world problems. Applications of computational science, in fact, can be found in many fields, such as physics, chemistry, biology, genetics, hydrodynamics, finance, engineering, economics, geophysics, mechanics, cosmology, and many others.
Three major problem domains can be defined in computational science: modeling, simulations, and optimization. Modeling and simulations concern the design and tuning of models that reflect natural systems and processes, in order to understand known events, such as earthquakes, cells lifecycle, drug efficiency, or predict future situations, like weather, tornados, or economic risks. Optimization methods aim to find the "best" solution among a set of available alternatives in known scenarios, such as engineering processes, protein design, disease detection.

**From *computational science* to *optimization*** Above all, problems in genomics are among the most difficult in computational sciences. They usually address the task of determining combinatorial properties of biological material, by comparing, discovering similarities and patterns in genomic sequences. There are several aspects in genomic problems that make them interesting targets for computational science. In particular, at an abstract level, genomic data (and, more in general, biological data) can be often approximated as sequences of elements, belonging to a specific alphabet; a sequence of such elements is identified as a string. Strings contain genetic material, DNA or RNA, that encodes biological instructions, for example to produce the proteins that regulate the life of organisms. Due to their form, these biological data necessitate of algorithmic methods to be treated, therefore the most viable approach is to combine computational and biological sciences. Drug and treatment discovery, disease modeling, protein structure prediction, genetic mapping, evolutionary tree reconstruction, are just few examples of computational problems in genomics.
As many of such problems are NP-hard, the study of improved techniques is necessary in order to solve them exactly, or at least with some guarantee of solution quality. In this scenario, mathematical optimization provides efficient methods to obtain optimal solutions and design desired behavior of

biological systems.
Optimization is a crucial task in the design of modern systems: identifying an optimal configuration, indeed, is a problem that arises in every area of sciences, ranging from industry-related problems, such as supply chain management or resources allocation, energy supply [PSP11], to micro-electronic design, telecommunication [SPP11], medicine, economics and finance. Optimization methods are designed to identify near-optimal solutions to real-world problems, where the notion of optimality is intrinsically related to the specific problem discussed. In general, finding an optimal solution is an intractable problem; although many results of mathematical analysis assure the presence of global optimizers for many problems, it is common to face real world applications where analytical methods are not applicable, due to the roughness of the search landscape or the difficulty in approximating derivatives. To overcome these limitations, a plethora of heuristic methods has been designed; in general, these approaches require minimal expert knowledge of the domain and ensure a good sampling of the solution space. Some of the most popular heuristic optimization methods are Genetic Algorithms, Simulated Annealing, Direct Search and Tabu Search.

This thesis is mainly focused on this domain, by addressing the study and design of optimization approaches for problems in genomics. In particular, two different classes of problems related to the analysis of genomic data are discussed, and new computational approaches are introduced. The problems addressed concern the selection and comparison of genomic sequences, and the design and selection of optimal probes to perform efficient hybridization experiments.

**Thesis contribution and organization** In order to settle the background of this thesis, we address some preliminary questions on optimization problems and methods in Chapter 2: after a formal definition of optimization problems, we focus our attention on the quality of the solutions and on the complexity of the problems. Despite feasible solutions might be "easily" found in some cases, the goal of optimization is the identification of the "best" solution among a set of available alternatives, which is a difficult task. In fact, the complexity of the problems makes brute-force approaches impracticable, therefore effective and efficient methods are required. In such context, we distinguish between exact and heuristic methods: exact approaches pro-

vide quality guarantee, but they are not always applicable; in such cases, heuristic methods offer a valid alternative to tackle optimization problems. Hybrid approaches take both the advantages of optimality guarantee by exact methods, and computational efficiency provided by heuristic strategies. For a more detailed discussion on such optimization strategies, we remand the reader to Chapter 2.

One of the main issues occurring in computational biology and bioinformatics concerns the localization of similar features in DNA or protein sequences. The *Closest String Problem* (*CSP*) represents one such problem, which consists in finding a string that is close to each of the strings in a given finite set. Such combinatorial optimization problem finds applications, for instance, in genetic drug target and genetic probes design [LLM+99], in locating binding sites [HHS90, SH89], and in coding theory [FL97, GJL99, Rom92]. Recently, the *CSP* problem has received increasing attention: several approximation algorithms have been developed [FL97, LLM+99, LMW02, MS08], and an integer programming formulation along with a branch-and-bound approach has been proposed [MLO+04]. Nevertheless, this latter approach presents exponential time complexity and memory explosion. Another approach to NP-hard problems consists in applying heuristic methods; these approaches do not guarantee to find an optimal solution, but in general, they are able to provide a good feasible solution, i.e. a solution with a value that is "close" to the optimum, in reasonable amount of time. Simulated Annealing, Genetic Algorithms [LHS05], hybrid methods [LHHW08], are heuristic approaches proposed in literature for the *Closest String Problem*.
To overcome the NP-hardness of the problem [FL97, LLM+99], we present two new approaches in Chapter 3: the first method is based on the Ant Colony Optimization metaheuristic [Dor92, FP10], the second approach combines a Simulated Annealing algorithm with a new heuristic for finding a good initial solution for the problem, which allows to speed up sensibly the convergence of the algorithm [PCP11].
To assess the effectiveness and robustness of the proposed methodologies, we compared our proposed approaches extensively with the integer-programming (IP) exact solution, and with other heuristic methods present in literature, both on artificial and real instances. Experimental results show that our approaches allow to locate good solutions for the problem and outperforms the heuristic methods proposed for the *CSP*.

Another important issue in genomics is the identification of targets, generally viruses or bacteria, in biological samples. Biologists can use hybridization experiments to determine whether a specific DNA fragment, that represents the virus, is present in a DNA solution. A probe is a segment of DNA or RNA, labeled with a radioactive isotope, dye, or enzyme, used to find a specific target sequence on a DNA molecule by hybridization. Selecting unique probes through hybridization experiments is a difficult task, especially when targets have a high degree of similarity, for instance in case of closely related viruses. The class of Probe Selection Problems involves the selection of a small set of probes to analyze a population of unknown clones; in particular, the *Non-Unique Probe Selection Problem* (*NUPS*, for short) concerns the selection of non-unique probes, that are those hybridizing to more than one target [POP11a].
Several approaches are largely known in literature for the *Non-Unique Probe Selection Problem*, ranging from deterministic [KRS$^+$04, RSP07] to heuristic methods [MPR07, NRWG10, WN07, WNGR08].
In Chapter 4, after discussing the techniques for the design and selection of hybridization probes, two new heuristic approaches are introduced for the NUPS problem; the first one, is a canonical Monte Carlo algorithm with a heuristic reduction. Starting from the results obtained by the Monte Carlo method, a new combinatorial optimization approach called Space Pruning Monotonic Search is proposed. The experimental results show that both of the presented approaches are suitable for the problem and, in particular, the Space Pruning Monotonic Search clearly outperforms the current state-of-the-art methods [POP11b].
Finally, in Chapter 5, we draw our conclusions.

Despite the promising results found by the methods proposed in this thesis for the *Closest String Problem* and the *Non-Unique Probe Selection Problem*, we are conscious there is still a lot of work on these subjects. Moreover, new emerging scenarios in biology and medicine, such as the design of custom drugs, the detection of diseases, the control of drug effects, require the design and development of computational methods and optimization approaches to be tackled. Therefore, this thesis represents only a first step in what should be a continual research towards the design of effective optimization methods for problems in biomedicine.

# 2
# Mathematical Optimization

*"Consider everything. Keep the good.*
*Avoid evil whenever you notice it."*

1 Thess. 5:21-22,
cited by A. Neumaier

Mathematical Optimization is an interdisciplinary branch of applied mathematics, related to the fields of Operations Research, Computational Complexity and Algorithm Theory. It can be informally defined as the science of finding the "best" solution from a set of available alternatives, where the notion of "best" is intrinsically related to the specific problem addressed.
Optimization problems are countless in everyday life: from the choice of the fastest way to get from home to office, to large scale problems such as the maximization of company profit or the minimization of the side effects of drugs, optimization is a crucial task in the design of modern systems. In particular, optimization problems arise in every area of sciences: engineering, microelectronics, telecommunications, biomedicine, genetics, proteomics, economics, finance, physics,
Unfortunately, for many of these problems, it is not known whether and how they can be solved exactly in a "reasonable" time. Optimization methods are designed to identify *near-optimal* solutions to these problems, where the notion of *optimal* is intrinsically related to the specific problem discussed.

In this thesis, we focus our attention on two real-world optimization problems that arise in biomedical research. We start by settling the general context and motivations of our study; in particular, in this chapter we address some general questions about optimization problems and their solutions, in

order to provide a background knowledge of the issues analyzed later.
We refer the reader to Chapter 3 and 4 for a detailed introduction and description of the problems studied.

## 2.1 Optimization Problems: what, how and how much

### 2.1.1 *What* is an optimization problem?

In the simplest case, the optimization process seeks to minimize or maximize the value of a function, defined as *objective*, by choosing integer or real variables from a defined domain of possible alternatives. Formally, an *optimization problem instance* can be stated as follows [PS98]:

**Definition 2.1.1** (Optimization Problem)**.** *Given a function $f : S \to \mathbb{R}$ from some set $S$ to the real numbers, the objective is to find an element $x_0$ in $S$ such that $f(x_0) \leq f(x)$ for all $x$ in $S$ ("minimization"), or such that $f(x_0) \geq f(x)$ for all $x$ in $S$ ("maximization").*

$S$ is called *feasible set*, and it represents the set of all the possibilities from which a feasible, or valid, solution can be built, i.e. a solution satisfying all the constraints imposed for the problem.
$f$ is the *objective function*, that evaluates the quality of feasible solutions by assigning a certain "quality measure", defined according to the specific problem analyzed.
The point $x_0$ is called a *globally optimal solution* to the given instance, that is a feasible solution minimizing/maximizing the objective function value.
Informally, we can distinguish between a *problem* and an *instance* of the problem: the pair $(S, f)$ is defined as an *instance* where, given the input data for the problem, a solution can be obtained if there is enough information available. A *problem* is a collection of instances. Therefore, while the instance represents a particular input to the problem and its corresponding solution, an optimization problem can be viewed as an infinite collection of instances, each of them characterized by a specific solution.
Given an algorithm $A$ for an instance $(S, f)$ of an optimization problem, $A$ is said to be *exact* if it always returns an optimal solution, and $A$ is said to be *efficient* if it runs in time polynomial on the size of its input.

Optimization problems can be divided in two categories: those involving *continuous* variable, and those with *discrete* variables. While there are more theoretical results in continuous than in discrete optimization, many real world problems involve the presence of discrete components. Combinatorial Optimization represents the subset of Mathematical Optimization where the solution domain is discrete or can be reduced to discrete [PS98]. This class of methods finds application in many research areas, including management sciences such as finance, marketing, supply chain, scheduling; in engineering, as communication and energy related problems. In particular, optimization has become an indispensable tool in many areas of biomedicine (e.g. X-ray crystallography, protein folding, epileptic seizure prediction, etc.), and it is frequently used for designing and modeling complex systems, which are abundant in biology.

### 2.1.2 *How* good is a solution?

In general, finding the global optimum to an instance of an optimization problem can be a very difficult task, and it is computationally intractable for many problems. Nevertheless, it is often possible to find a solution that is the *best* within a specific subset of the solution space, defined as a *neighborhood*. This solution represents a *local optimum*. The search of an arbitrary local optimum is a task addressed by local search methods. Shortly, a local search starts from an initial solution and iteratively modifies that solution into a better one, by performing a move, i.e. a small perturbation, on it. Such move identifies a *neighbor* solution.
Two important issues in local search methods are the quality of the solution found, and the complexity of the local search heuristic, i.e. how fast the local optima can be found. Usually, the best choice is a tradeoff between quality and computational complexity: whereas the larger is the neighborhood, the better will be the solution found, it might be computationally intractable to compute it. Therefore, the design of a local search heuristic involves the choice of a "good" neighborhood, that is a neighborhood large enough to contain good solutions, and on the other hand, small enough to ensure an efficient search.

#### 2.1.2.1 Neighborhood

Local search is based on the concept of *neighborhood*. Given a feasible point $x \in S$, a neighborhood can be informally defined as a set $N(S)$ of points that are close, in some sense, to the point $x$, for example because they can be computed starting from $x$, or they share a significant part of their structure with $x$. Formally, a neighborhood can be defined as follows.

**Definition 2.1.2** (Neighborhood)**.** *Given an instance $(S, f)$ of an optimization problem, a neighborhood is a mapping*

$$N : S \to 2^S$$

*defined for each instance, where $2^S$ denotes the powerset $\{V|V \subseteq S\}$.*

The neighborhood function $N$ specifies, for each solution $x$, a set $N(x) \subseteq S$, which represents the neighborhood of $x$. Therefore, we say that a solution $x'$ is a neighbor of $x$ if $x' \in N(x)$.

A local search algorithm searches through the solution space by iteratively moving from the candidate solution to one of its neighbors. This process can be viewed as a walk through the *neighborhood graph*, where the vertices represent the solution points, and the arcs connect the neighbors.

**Definition 2.1.3** (Neighborhood Graph)**.** *Given an instance $(S, f)$ of a combinatorial optimization problem and a neighborhood function $N$, the neighborhood graph is a directed node-weighted graph $G = (V, E)$, where the node set $V$ is the set of solutions $S$, and the arc set $E$ is defined such that $(i, j) \in E$ if and only if $j \in N(i)$.*

In such graph, the weight of a node $j$ represents the cost of the corresponding solution. A solution $j$ is said to be *reachable* from a solution $i$, if the neighborhood graph contains a path from $i$ to $j$, that is, there exists a sequence of solutions $x_i, x_{i+1}, ..., x_j$, for $j \geq 1$, and $x_{l+1} \in N(x_l)$, $i \leq l < j$.

**Definition 2.1.4** (Neighborhood-optimal solution)**.** *A solution $\hat{x} \in S$ is locally optimal with respect to the neighborhood N, or N-optimal, if for all $x \in N(\hat{x})$, $f(\hat{x}) \leq f(x)$.*

The neighborhood function may, or may not, generate the globally optimal solution. When the neighborhood function is able to generate the global optimum, starting from any initial feasible point, it is called *exact*.

**Definition 2.1.5** (Exact Neighborhood)**.** *Given an optimization problem with feasible set $S$ and a neighborhood $N$, if each locally optimal solution $x_0 \in S$ with respect to $N$ is also globally optimal, we say that the neighborhood $N$ is exact.*

### 2.1.3 *How (much)* difficult is the problem?

Despite the fact that many real-world problems can be modeled as combinatorial optimization problems, they are often very difficult to solve. In order to develop a measure of the difficulty of a problem, we have to quantify the resources needed to find a feasible solution. However, the choice of physical measures, such as run-time or memory requirements, are irrelevant, since they depend on the computing platform and on the specific algorithm implemented to solve the problem. Moreover, the time needed to solve a problem instance depends on certain properties of the instance itself, such as its size: larger instances, for example, require more time to be solved. Thus, the time required to solve a problem has to be a function of the size of the instance.
Computational complexity analyzes the inherent intractability or tractability of such problems, and classifies them into complexity classes according to their computational hardness. Specifically, if we consider computational time, two main classes can be identified, that are P and NP.
The first complexity class contains all the problems that can be solved by polynomial-time algorithms. This class is denoted by P, that stands for polynomial time. P can be considered as the class of tractable problems, that can be solved efficiently.
NP indicates the class of nondeterministic polynomial problems, that is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine.
It is not known whether P=NP, though it is widely believed that there are problems in NP which are not in P. Some such problems have a property known as NP-*completeness*. NP-complete problems are considered intractable, and only approximate solutions are known for them. In particular, if an NP-complete problem could be solved in polynomial time, thus all NP-complete problems can be solved in polynomial time, with the consequence that P=NP. Formally, NP-completeness is defined as follows:

**Definition 2.1.6** (NP-completeness)**.** *A decision problem $C$ is NP-complete if $C$ is in NP, and every problem in NP is reducible to $C$ in polynomial time.*

Specifically, a problem $A$ is said to be "reducible in polynomial time" to a problem $B$ if, for each instance $a$ of $A$, it is possible to build an instance $b$ of $B$ in polynomial time, such that the two instances have the same truth values.
A problem satisfying the second condition of NP-*completeness*, but not necessarily the first one, is said to be NP-*hard*. Informally, an NP-hard problem is "at least as difficult as" every other NP-complete problem.

Since many problems in combinatorial optimization are NP-hard, it is fundamental to explore techniques for dealing with NP-completeness. In the next section, we provide an introduction of the main approaches used to solve NP-hard problems.

## 2.2 How to solve it?

The available techniques for solving combinatorial optimization problems can be roughly classified into two main categories: *exact* and *heuristic* methods. While exact methods allow to find solutions with a guarantee of optimality, the run-time increases dramatically with the instance size. Heuristic approaches, on the other side, sacrifice optimality guarantees in order to find solutions in a more efficient way, by spending only a "reasonable" amount of time to compute them. The combination of exact and heuristic approaches defines the category of *hybrid methods*, which bring the advantages of both techniques.

### 2.2.1 Exact Methods

Exact methods have the advantage of guaranteed quality of the solution found, but the computational demand of such methods can be exponential in the worst case. Exact methods can be classified into the following categories [Pin02]:

- *Adaptive stochastic search methods*, based on random sampling in the feasible set. These approaches can implement some strategies to improve the random sampling, such as adaptive search strategy adjustments, sample clustering, deterministic solution refinement options, statistical stopping rules [Zhi91].

- *Bayesian search algorithms*, where a stochastic model is constructed to guide the search phase [Moc97, Pel05].
- *Branch and Bound algorithms*, that are tree pruning techniques, based on three main components: the selection of the node to process, bound calculation, and branching. These operations are applied iteratively to the collection of active subsets within the feasible set [Neu90, HW04].
- *Enumerative strategies*, which enumerate all the possible solutions to the problem [HT94].
- *Homotopy and trajectory methods*, that aim at visiting all stationary points of the objective function [For95].
- *Integral methods*, that approximate the level sets of the objective function to determine its essential supremum [ZZ95].
- *"Naive" approaches*, such as simultaneous grid search and pure random search [Zhi91].
- *Relaxation strategies*, where the optimization problem is replaced by a sequence of relaxed sub-problems, easier to solve [HT94].

In general, finding an optimal solution is an intractable problem; although many results of mathematical analysis assure the presence of global optimizers for many problems, it is common to face real world applications where exact methods are not applicable, due to the roughness of the search landscape or the difficulty in approximating derivatives. To overcome these limitations, a plethora of heuristic methods has been designed; in general, these approaches require minimal expert knowledge of the domain and ensure a good sampling of the solution space.

### 2.2.2 Heuristic Methods

Etymologically, *heuristic* comes from the Greek word ευρισκειν *(euriskein)*, that means "to discover", "to find". In fact, although heuristic methods do not guarantee to find an optimal solution, they should be able to "discover" a "good" solution in a shorter period of time than exact approaches. In particular, *metaheuristics* can be defined as upper level methodologies for solving a very general class of computational problems, as the Greek suffix μετά *(meta)*

suggests. Metaheuristics include many methods, such as Simulated Annealing [KGV83], Tabu Search [GL98], Variable Neighborhood Search [MH97], and several Evolutionary approaches and Memetic algorithms.

Since heuristics cannot guarantee the solution optimality, we might wonder why these approaches have became widely used for addressing complex optimization problems. In many cases, in fact, heuristics are preferable to exact methods, for instance when such methods are not available, or when an exact approach is available but it is computationally unfitting, due to the excessive time or storage requirements; heuristics are also used to improve the performance of an optimizer, since they are able to provide starting solutions or a pruned search space. Additionally, due to their relatively simple features, they can be easily implemented and therefore used to model complex problems with a more pragmatic approach [ZE81].
In general, a "good heuristic" should be characterized by reasonable computational efforts, such as reasonable storage requirements and computing times, that do not grow exponentially as the problem size increases. Other important features of good heuristics are simplicity, accuracy and robustness, i.e., the method should not be sensitive to changes in parameters; it should produce multiple solutions by perturbing the current point; it should implement good stopping criteria, in order to avoid "stagnation", and having good average performance [ZE81].

Due to their computational power, a large number of heuristic methods have been proposed to solve real-world problems in several areas. Heuristic approaches can be grouped into some main categories [Pin02]:

- *"Globalized" extension of local search methods*, characterized by a preliminary global search, such as a grid or random search, followed by a local search phase [VOR99].

- *Evolution strategies*, that represent an abstraction of the biological evolution of living organisms, by implementing the "survival of the fittest" principle. In evolutionary approaches, in fact, a population of individuals is evolved, by means of natural selection and sexual reproduction processes. Individuals with better fitness have higher probability to reproduce and survive. Numerous variants of this general strategy

can be constructed, according to different evolutionary "game rules" [GH88, Gol87, Mos89, BGH90].

- *Simulated Annealing*, based on the physical analogy with the thermodynamic cooling process of crystal structures, that reach a stable configuration characterized by minimal potential energy. This principle is applied to solve discrete and continuous optimization problems [MRR$^+$53, MU49, KGV83, HJJ03, RSV91, Haj88, LM86].

- *Tabu Search*, that implements the idea of forbidden moves to points in the search space that have been already visited in the previous steps. This strategy aims to explore new regions of the search space, with the goal of finding a global optimum [GL98, VOR99].

- *Approximate convex underestimation*, that tries to estimate the large scale, "overall" convexity characteristics of the objective function, by performing direct sampling of the search space [Pin02].

- *Continuation methods*, that first transform the objective function into a "smoother", simpler function having fewer local minima, and then use a local minimization procedure to trace the minimizers back to the original function [For95, Pin02].

- *Sequential improvement of local optima*, such as tunneling, deflation, and filled function methods, that usually operate on adaptively constructed auxiliary functions, to assist the search towards gradually better solutions [LG85, Pin02].

### 2.2.3 Hybrid Methods

Hybrid methods combine the advantages of both exact and heuristic approaches: they mainly aim at providing solutions with optimality guarantee in shorter time. Hybrid methods can be mainly classified into two categories: *collaborative combinations* and *integrative combinations* [PR05].
The first class defines the algorithms that exchange information, but are implemented as separate methods: exact and heuristic methods can be executed in sequence, in parallel or can be intertwined. Such approaches have been broadly proposed in literature, for instance evolutionary algorithms have been combined with branch-and-bound techniques, and heuristic strategies

have been applied to provide a pruning of the search space and speed up the computation of optimal solutions.
When exact methods and heuristics are combined into integrative combinations, one technique is a subordinate embedded component of the other. This is the case where metaheuristics are used within an exact method to determine bounds and incumbent solutions in branch-and-bound approaches, or the case of heuristic column generation in branch-and-price algorithms. Another example of integrative combinations is the exploration of neighborhoods in local search heuristics by means of exact algorithms.

In the following, we propose several mathematical formulations, heuristic and metaheuristic methods for addressing biomedical and engineering problems. Specifically, we implement a Monte Carlo and a Simutated Annealing algorithms, along with a Mesh Adaptive Basin Hopping method. Additionally, in consideration of the effectiveness and quality guarantee provided by hybrid methods, we implement a hybrid approach in Chapter 4 to address the *Non-Unique Probe Selection Problem*. The computational power of the proposed methods is extensively explored, and experimental results are provided in order to compare our approaches with the state-of-the-art.

# 3

# String Selection and Comparison Problems

*"Share our similarities, celebrate our differences."*

M. Scott Peck

## 3.1 Introduction

Starting from the introduction of the first chain-termination method for DNA sequencing in 1977 by Sanger *et al.* [SNC77], the beginning of a new era of advancements in molecular biology, chemistry, and computational science has been marked [Bou10]. DNA sequencing, in fact, includes several methods and technologies to map out the sequence of the nucleotides present in a strand of DNA. Specifically, DNA sequencing results in a sequence of DNA components called primary structure, consisting of four symbols $A$, $C$, $G$, and $T$, corresponding to the nucleotide bases of a DNA strand, adenine, cytosine, guanine, and thymine, which encode the genetic information represented by the DNA sequence.

Due to the growing amount of data made available through sequencing techniques, the process of manually analyzing DNA sequences has became impracticable. This scenario led to the development of new scientific branches, such as computational biology and bioinformatics. These biological sciences study and develop new approaches for treating genomic data, and for addressing specific problems related to the recognition of features and functions within biological samples.

One of the main issues occurring in computational biology and bioinformatics

is to locate similar features in DNA or protein sequences; this information can be used to detect potentially important regions within them [GJL99]. The recognition of similarities and analogies in genomic samples finds applications in the design of genetic drug and genetic probes [LLM+99], and in locating binding sites for detecting disease risk [SH89, HHS90]. Other applications concern the field of data compression [GS85], and coding theory [GJL99, FL97], where one is interested in determining the best way to encode a set of messages [Rom92].
In molecular biology, in particular, the identification of common regions within DNA or amino acid sequences provides relevant insights on the function of proteins and genes: high similarity among the sequences, in fact, usually involves a structural or functional affinity. This information can be used, for instance, to locate genes whose mutation is involved in genetic diseases, by detecting the differences between genes of healthy and unhealthy individuals.

Specifically, in order to address the task of discovering analogies and/or differences in genomic samples, some similarity measures have to be defined. To measure the number of similarities between two strings, two distance metrics are commonly adopted: the *Levenshtein* or *edit distance* [Lev66], and the *Hamming distance* [Ham50]. The edit distance between two strings is defined as the minimum number of edits, that are insertions, deletions and substitutions, required to transform one string into the other. The Hamming distance represents a special case of the edit distance, where only mismatches between strings are considered.
When the two strands of DNA bind together according to the Watson-Crick base pairing [WC53], they are said to *hybridize*, and if this matching is exact, one sequence is said to be the *reverse complement* of the other. Specifically, given the four nucleotide bases, adenine (A) and thymine (T) form a complementary pair, as do cytosine (C) and guanine (G). We describe this pairing by using the notation $\overline{A}$=T, $\overline{T}$=A, $\overline{C}$=G, and $\overline{G}$=C. For example, the Watson-Crick complement of the sequence AACATG is represented by the sequence $TTGTAC$.
The differences discovered between the sequences can be classified into two main categories: *substitutions*, where a base in one strand does not pair with the base on the other strand, and *gaps*, characterized by the presence of at least one extra base in one strand. For instance, A**A**CATG and $T$**G**$GTAC$

present a substitution at the second base of the second sequence, where the thymine is replaced by a guanine base, which does not match with the adenine of the first sequence. If we consider now the sequences AACATG and $T - GTAC$, we can see that a gap is present in the second sequence. Since gaps are more destabilizing than substitutions [LLM+99], the use of the Hamming distance, that involves only substitutions, is preferable as distance metric to the edit distance, which considers also gaps.
The Hamming distance metric appears in several biological applications, and especially in those related to the discovery of consensus sequences or the use of its reverse complement [LLM+99]. Some applications are the design of diagnostic probes, drug design, and creation of PRC primers [Lan04, Bou10].

**Design of Diagnostic Probes** The task of designing a string that is able to represent a set of known sequences and, at the same time, is easily distinguishable from another set, is a problem arising in the diagnosis of viruses and bacteria in host organisms. In particular, probes are used to discover the presence of viruses and bacteria in biological samples. A probe is a strand of DNA or RNA opportunely treated to easily detect the presence of a target, for instance by making it fluorescent or radioactive. Hence, given a set of DNA sequences representing the virus or the bacteria, and a host, the problem is to discover a sequence that occurs in DNA sequences, and does not appear in the host. This problem involves the design of the probe sequence, that has to be as close as possible to the sequences to detect, and, on the other hand, as far as possible from another set of sequences.

**Drug Design** Another important application of string selection concerns the drug design. Here, given a set of sequences of orthologous genes[1] from a group of closely related pathogens, and a host, the goal is to identify a sequence that is highly conserved in all of the pathogen sequences, but that is not present in the host [LLM+99]. In fact, the conserved region might encode relevant biological information, since it seems resilient to mutations. This information can be exploited to identify the chemical components that bind the conserved region, in order to create new effective therapies.
In antisense drug design, the same principle is used to create drugs that inhibit the production of the protein related with the disease, but do not

[1]We recall that orthologous are genes in different species that evolved from a common ancestral gene, and which encode proteins with the same function in different species.

interfere with other proteins [LLM+99]. Specifically, antisense therapies focus on impeding the production of proteins that cause the disease: antisense binds mRNA to prevent that the genetic code related to the disease will be read by the ribosome, which is responsible in assembling proteins based on the instructions carried by mRNA.

**Primers Design for Polymerase Chain Reaction** Polymerase chain reaction (PCR, for short) is a technique adopted in molecular biology for amplifying a portion of DNA in many copies. PCR has many applications, such as DNA cloning for sequencing, functional analysis of genes, forensic identification, disease diagnosis. The PCR process requires the selection of two primers, that are small fragments of DNA, hybridizing within a specific region. The selection of primers is a complex task, and it affects the PCR amplification process. In particular, the selection of primers that are able to amplify several regions simultaneously can be viewed as a sequence problem where the goal is to determine the maximum number of mismatches allowed in the hybridization process [Bou10].

The class of problems addressing the recognition of similar characteristics or differences within biological sequences has been defined by several names, such as *distinguishing string selection* [LLM+99], *consensus patterns* [HHS90], *Hamming center problems* [GJL99] and *motif detection problems* [Bou10]. Specifically, two main problems fall into these categories: *Farthest String* and *Closest String* problems.

The *Farthest String Problem* [LLM+99, Lan04, Bou10] informally defines the problem where a pattern does not occur in a set of strings, and has applications in the analysis of genomic data. Such problem has been proved NP-hard, therefore it is unlikely to solve in polynomial time, unless P=NP. The *Closest String Problem* [LLM+99, Lan04, Bou10], contrary to the previous class, defines the task of finding a pattern that, with some errors, occurs in a specific set of strings. In particular, it consists in finding a string with minimum Hamming distance from the strings of a given finite input set.

To overcome the NP-hardness of the problem, we developed two approaches which are presented in this chapter: the ANT-CSP algorithm, based on the

*Ant-Colony Optimization* metaheuristic [FP10], and a Simulated Annealing approach combined with a new heuristic used for finding a good initial solution for the problem [PCP11]. To assess the effectiveness and robustness of the proposed methods, we extensively compare them with the other approaches presented in literature, both on artificial and real instances. Experimental results show that our approaches allow to locate good solutions for the problem, and outperform the other heuristic methods proposed.

## 3.2 The Closest String Problem

In this section, we define the *Closest String Problem* (*CSP*, for short), which informally denotes the task of finding a string that presents many similarities with a specific set of sequences.
The *CSP* problem finds applications in many fields, such as genetic drug target and genetic probes design [LLM+99], in locating binding sites [SH89, HHS90], and coding theory [GJL99, FL97, Rom92]. For a detailed discussion on the biological relevance of string selection problems, we refer the reader to Section 3.1.

A precise definition of the *CSP* is given next. To begin with, we need to recall some notations. Given a string $s$ over a finite alphabet $\Sigma$, $|s|$ and $s[i]$ denote the length of $s$ and the $i$-th character of $s$, respectively. We recall that the Hamming distance between two strings $s$ and $t$ having equal length is the number of positions at which $s$ and $t$ differ.

**Definition 3.2.1** (Closest String Problem)**.** *Let $S = \{s_1, s_2, ..., s_n\}$ be a finite set of $n$ strings, over an alphabet $\Sigma$, each of length $m$.*
*The Closest String Problem for $S$ is to find a string $t$ of length $m$ over $\Sigma$, that minimizes the Hamming distance $d_c$ such that, for any $s \in S$,*

$$d_c(s, t) \leq d_c.$$

Let us consider the following example: given four strings on a binary alphabet $\Sigma = \{0, 1\}$, $S = \{001000, 111000, 011011, 010101\}$, the string $t = 011001$ is a *closest string* for $S$; in fact, $\max_{j=1}^{4} d_c(s_j, t) = \max\{2, 2, 1, 2\} = 2$.

The *Closest String Problem* can be viewed as a special case of a more general problem, defined as *Closest Substring Problem*, (CSSP, for short) whose formal definition is the following:

**Definition 3.2.2** (Closest Substring Problem)**.** *Let $S = \{s_1, s_2, ..., s_n\}$ be a finite set of n strings, over an alphabet $\Sigma$, each of length at least $m$.*
*The Closest Substring Problem for $S$ is to find a string $t$ of length $m$ over $\Sigma$, that minimizes the Hamming distance $d_c$ with every substring $z$ of $s_i$ of length $m$, $z = s_i[h, l]$, $l = m + h - 1$ and $1 \leq i \leq n$:*

$$d_c(s, z) \geq d_c.$$

The *Closest String Problem* and the *Closest Substring Problem* are NP-hard; in particular, Frances and Litman have proved the NP-hardness of the *CSP* for binary codes, by considering an equivalent problem in coding theory [FL97]. Then, Lanctot *et al.* have shown that even in the case of alphabets with more than two characters the problem is NP-hard [LLM$^+$99]. Therefore, no polynomial-time solution is possible, unless P = NP.
Since the *Closest String Problem* represents a special case of the *Closest Substring Problem*, it follows that also the *CSSP* is NP-hard [LLM$^+$99].

Recently, the *CSP* and *CSSP* have received increasing attention. We discuss some past results concerning such problems in the following section.

## 3.3 State-of-the-art Methods

The first integer-programming (IP, for short) formulation for the *CSP* has been proposed, independently, in [BDLRP97] and [LMW02]. In [MLO$^+$04], three IP formulations are presented for the problem along with a heuristic: the *CSP* is first reduced to an integer-programming problem, and then a branch-and-bound algorithm is used to solve it.
Specifically, an instance of the *CSP* consists of a finite set $S = \{s_1, s_2, ..., s_n\}$ of strings such that $|s_i| = m$. By $s_i[j]$, we denote the $j$-th position of the string $s_i$, for $1 \leq i \leq n$ and $i \leq j \leq m$. The goal of the problem is to find a string $t \in \Sigma^m$ such that $\max_i d_c(s_i, t)$ is minimized, where $d_c$ represents the Hamming distance.
Instead of working directly on strings, an injective transformation $\pi$ is applied, which maps each character $c$ into an integer $\pi(c)$, and any string $s = (s[1], s[2], ..., s[m])$ into a sequence $\pi(s) = (x[1], x[2], ..., x[m]) \in \mathbb{Z}$ of the same length, such that $x[k] = \pi(s[k])$, for $k = 1, ..., m$.
For instance, given the alphabet $\Sigma = \{a, ..., z\}$, let us consider the "canonical" transformation $\pi(a) = 1, \pi(b) = 2, ..., \pi(z) = 26$. Then, the string $s =$ *differ* is mapped into $\pi(s) = (4, 9, 6, 6, 5, 18) \in \mathbb{Z}^6$.

Given a set $X = \{x_1, ..., x_n\}$ of strings of the same length $m$, $V_k$ represents the set of the $k$-th characters of the strings $x_i$, that is $V_k = \{x_1[k], x_2[k], ..., x_n[k]\}$, for $i = 1, ..., n$ and $k = 1, ..., m$.
Additionally, the binary variables $v_{j,k}$ are defined, which characterize a candidate solution $t$ as follows:

$$v_{j,k} = \begin{cases} 1 & \text{if } t[k] = j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

for $j \in V_k$, $k = 1, ..., m$.
Therefore, the integer-programming formulation for the *CSP* is defined as follows [BDLRP97, LMW02, MLO+04]:

$$\min d_c \tag{3.2}$$

subject to

$$\sum_{j \in V_k} v_{j,k} = 1 \qquad k = 1, ..., m \tag{3.3}$$

$$m - \sum_{j=1}^{m} v_{x_i[j],j} \leq d_c \qquad i = 1, ..., n \tag{3.4}$$

$$0 \leq v_{j,k} \leq 1 \quad j \in V_k, k = 1, ..., m \tag{3.5}$$

$$d_c \in \mathbb{Z}_+ \tag{3.6}$$

The objective function (3.2) minimizes the Hamming distance $d_c$, whereas (3.3) and (3.5) guarantee that exactly one character in $V_k$ is selected. Inequalities (3.4) say that if a character in a string $s[i]$ is not chosen for a solution $t$, then such character will contribute to increase the Hamming distance between $t$ and $s[i]$. Finally, constraints (3.6) force the distance $d_c$ to assume a non-negative value.
The above formulation (3.2)-(3.6) is the most effective from a computational point of view, since it involves fewer variables and constraints [MLO+04, CWB11]. Nevertheless, the approach proposed in [MLO+04] is not always efficient and has an exponential-time complexity. Moreover, the branch-and-bound technique leads easily to memory explosion.
In order to speed-up the solution search, a parallel version of the heuristics presented in [MLO+04, MOP05] is developed in [GMPV08]. A new formulation for the *CSP* is proposed in [CWB11], although experimental results prove that it is not as effective as the one proposed in [BDLRP97, LMW02] and [MLO+04].

Due to the limited efficiency of exact methods in solving large instances of the problem, a plethora of non-exact approaches have been proposed in literature. In general, we can distinguish between two classes of non-exact methods for solving optimization problems: *approximation algorithms* and *heuristics*. Approximation algorithms guarantee a bound on the optimal solution, whereas heuristic strategies do not. Several approximation algorithms have been proposed for the *CSP* problem: two $(4/3+\epsilon)$-polynomial time approximation algorithms have been developed in [GJL99, LLM$^+$99].
To overcome the NP-hardness of the problem, a fixed-parameter algorithm with running time $\mathcal{O}(nm + nd^{d+1})$ has been developed in [GNR01, GNR03], where $d$ is a parameter that represents the maximum Hamming distance allowed. The underlying idea of this approach is to study the parameterized complexity of the problem, under the assumption that either $d$ or the number of input strings $n$ are small. Plainly, for large values of $d$, such approach becomes prohibitive. A better approximation for the *CSSP* is provided in [LMW99], that gives a ratio $2 - \frac{2}{2|\Sigma|+1}$, where $|\Sigma|$ is the cardinality of the alphabet. When $|\Sigma|$ is large, the improvement provided by this approach is not remarkable. For this reason, a new polynomial time approximation scheme for the *CSSP* is presented in [Ma00], that combines a random sampling procedure with the method presented in [LMW99].
Based on the previous approaches, new polynomial-time approximation algorithms for the *CSP* and *CSSP*, having approximation ratio $1 + \epsilon$, for any small $\epsilon$, have been developed [LMW02]. A further improvement is presented in [MS08], where an algorithm that finds the optimal solution for *CSP* with running time $\mathcal{O}(mn + nd(16|\Sigma|)^d)$ is proposed.
The complexity results are improved in [WZ09], where a $\mathcal{O}(mn + nd(|\Sigma - 1|^d 2^{3.25d})$-time fixed parameter algorithm for the *CSP* is presented. Recently, a polynomial-time approximation algorithm having a new time bound that outperforms the other approaches has been presented [CMW11].
However, the running time of the approximation algorithms presented for the *CSP* and *CSSP*, makes them of theoretical importance only.
In terms of parameterized complexity, the main result for the *Closest Substring Problem* is that it cannot be solved in polynomial time, even when the distance parameter is fixed. This is expressed by showing that the *CSSP* belongs to the complexity class of W[1]-hard problems [FGN02, MOP05].

Since heuristics are usually able to provide good solutions in a reasonable amount of time, a plethora of heuristic approaches have been proposed for the *CSP*. *Genetic Algorithms* (GA, for short) [Hol92, GH88, BGH90] are among the most known metaheuristic methods; they mimic the natural evolutionary process by evolving a population of solutions. The first GA for the *CSP* has been proposed in [MMH03]; such paradigm has been successively adopted in [LHS05, Jul09]. A *Simulated Annealing* approach has been presented in [LHS05], and a hybrid algorithm has been implemented in [LHHW08], which combines both the Genetic and the Simulated Annealing approaches, though limited only to binary alphabets. Finally, a *Memetic Algorithm* has been proposed in [BM10], which integrates a local search strategy with an evolutionary method. In a more recent work [Tan11], a heuristic strategy based on a Lagrangian relaxation and a *Tabu Search* method has been proposed for the *Closest String Problem*.
In [LLHM11], an exact algorithm for the case of three strings with alphabet size equal to two is presented; however, this case is only of theoretical interest. A new approach has been introduced in [KK10], where the *CSP* is addressed as a constraint satisfaction problem. Despite the innovation proposed by this method, the results are claimed just for short instances of the problem, that do not represent a real application.

Several heuristic approaches have been proposed in literature also for the *Closest Substring Problem* [BT02, PS00, PRP03]. In particular, two exact algorithms for small values of $d$ and $n$ have been implemented [Mar08]. Since, for real instances, the values of $n$ and $d$ are usually large, these approaches are characterized by only theoretical relevance.

## 3.4 Ant-Colony Optimization for the Closest String Problem

### 3.4.1 An Overview of the Ant-Colony Optimization Metaheuristic

The *Ant-Colony Optimization* metaheuristic (*ACO*, for short) [DS02, DCG99, Dor92] is a multi-agent approach particularly suited for hard combinatorial optimization problems. In fact, this method covers two main application fields: NP-hard problems, whose best known solutions have exponential-time

worst-case complexity, and shortest path problems, in which the properties of the problems graph representation can change over time, concurrently with the optimization process. As the *CSP* problem is NP-hard, and searching a closest string can be viewed as finding a minimum path into the graph of all feasible solutions [MLO+04], it is natural to apply the *ACO* heuristic to the *Closest String Problem*. This is what we do next.

#### 3.4.1.1 Ant-Colony Optimization and Swarm Intelligence

The *Ant-Colony Optimization* metaheuristic is a member of a broader class of optimization methods, known as *Swarm Intelligence* (SI, for short). Swarm intelligence represents the collective behavioural intelligence of insect colonies, exploited to perform several tasks, such as nest construction, food transportation, task partitioning. SI provides an alternative way to design intelligent systems, where the notions of autonomy and distributed activities replace the concepts of control and centralization [BDT99]. In fact, in a colony of insects, each individual is not required to carry out all the tasks needed for the survival of the entire colony, but it is specialized in some of them; such partition of the work allows simultaneous activities, therefore it leads to a more efficient organization than performing sequential tasks. A key feature in insect colonies is their *self-organization*: individual behaviours emerge autonomously within the colony, without the guidance of any controller; such behaviours originate from local information, there is no global knowledge of the entire system. Specifically, the self-organization mechanism relies on four basic components: *positive feedback*, which promotes the most popular behavioural patterns within the colony; *negative feedback*, that serves as a regulatory control mechanism to balance the system; *amplification of fluctuations*, that supports exploitation and exploration through trials and errors; and *multiple interactions among single agents*.
The self-organization mechanism affects not only insect colonies, indeed it can be applied to design complex intelligent systems: a colony of insects can be thought of as a decentralized system for problem solution, based on the interaction of simple agents. In fact, insect colonies are extremely suitable to provide efficient and flexible solutions to complex problems.

Another important feature of SI is the *stigmergy*. Within a colony, two kinds of interactions can be detected: direct interactions, such as visual contact, antennation, chemical contact, and indirect interactions. The latters are more subtle: two individuals indirectly interact when one of them mod-

ifies the environment and the other responds to the new environment at a later time [RM04]; this form of indirect communication is used to coordinate the activities of the colony, and it is referred to as stigmergy.

#### 3.4.1.2 Real and Artificial Ants

*ACO* is modeled on the principles of Swarm Intelligence. *ACO* was firstly proposed by Dorigo as an innovative approach to the Traveling Salesman problem [Dor92].
Specifically, the *ACO* metaheuristic represents a transposition of real ants behaviour to artificial intelligence, and it has been inspired by the observation of real ant colonies, where the behaviour of each single ant is directed to the survival of the whole colony [DCG99]. In particular, in his analysis Dorigo pointed out the foraging behaviour of ants: when a new food source is found, ants search for the shortest and easiest way to return to nest. While walking from food sources to the nest, and vice versa, ants deposit on the ground a chemical substance called *pheromone* [DCG99]. Ants can smell pheromones and, when choosing their way, they select with higher probability paths marked by stronger pheromone concentrations. It has been proved that pheromone trails make shortest paths to emerge over other paths [DAGP90], due to the fact that pheromone density tends to be higher on such paths.
In view of all these facts, artificial ants are modeled on the behaviour of real ants.
As for real ant colonies, *ACO* is based on a population, or colony, of concurrent and asynchronous entities globally cooperating to find a good solution to the task under consideration; although each ant is able to find a feasible solution to the problem, high quality solutions are the result of the cooperation among the population.
Artificial ants modify some aspects of their environments: real ants release pheromone trails while walking; similarly, artificial entities release artificial pheromone trails, that are numeric information locally stored in the problem's state they visit. Such stigmergic communication allows to modify the perception of the environment within the colony. As real pheromones evaporate over time, usually *ACO* implements a pheromone evaporation mechanism; from a computational perspective, such strategy prevents the stagnation of the algorithm into a local optimum.
Both artificial and real ants share the goal of finding the shortest path from

an origin, that is the nest, to a destination, that is the food source. The notion of shortest path can be thought of as the solution having minimum cost. Ants move to adjacent problem states; this means they can not "jump" from a state to another, but just walk through adjacent locations.
Finally, artificial ants adopt a myopic and stochastic state transition policy: as the real ones, they take stochastic decisions to choose the next state to move to, by using only local information and without any prediction on the future states.
Despite these similarities, there are some differences between real and artificial ants: artificial ants move in a discrete world, and their moves consist on transitions from discrete to discrete states; artificial ants have an internal state to store their previous actions, and their timing in laying pheromone might implement a different policy than real ants.

In analogy with the real behaviour of ant colonies, therefore, *ACO* applies pheromone trails and social behaviour concepts to solve hard optimization problems. In short, *ACO* algorithms work as follows: a set of asynchronous and concurrent agents, a colony of ants, moves through the states of the problem. To determine the next state, ants apply stochastic local decisions based on pheromone trails. Ants can release (additional) pheromone into a state, while building a solution, and/or after a solution has been built, by moving back to the previous visited states.
In an elitist strategy, only the ant that has produced the best solution is allowed to update pheromone trails. In general, the amount of pheromone deposited is proportional to the quality of the solution built. To avoid a too rapid convergence towards suboptimal regions, *ACO* algorithms include another mechanism for updating pheromone trails, namely, pheromone evaporation. This consists in decreasing over time the intensity of the components of pheromone trails, as pheromone density tends to increase on shortest paths. Thus, the evaporation mechanism limits premature stagnation, that are the situations in which all ants repeatedly construct the same solutions, which would prevent further explorations of the search space.

#### 3.4.1.3 Convergence of ACO Metaheuristic

In order to assess the convergence of *ACO* metaheuristic, special classes of *ACO* algorithms have been broadly studied. In particular, two kinds of convergence have been proposed by Dorigo [DB05]: *convergence in value*, that

considers the probability the algorithm has to generate an optimal solution at least once, and *convergence in solution*, concerning the probability that the $ACO$ algorithm reaches a state which keeps generating the same optimal solution. A proof for the convergence in value and in solution of a special class of $ACO$ algorithms has been provided in [SD02], which considers a positive lower bound $\tau_{min}$ to pheromone values, in order to prevent the probability to generate any solution becomes zero. In particular, it has been proved that for any small constant $\epsilon > 0$ and for a sufficiently large number of algorithm iterations $t$, the probability of finding an optimal solution is given by $P^*(t) \geq 1 - \epsilon$, and it tends to 1 for $t \to \infty$.
While convergence proofs give theoretical results on relevant properties of the algorithms, they usually do not provide any practical advice for implementing efficient algorithms. An important practical result of the convergence of $ACO$ metaheuristic, is that $ACO$ suffers of deception [BD04]. The notion of deception has been introduced to describe problems which are misleading for Genetic Algorithms [Gol87]; example of deceptive problems are $n$-bit trap functions [DG93], that involve the presence of points corresponding to suboptimal solutions and characterized by large basins of attraction, or points with relatively small basins of attraction that correspond to optimal solutions. Therefore, Genetic Algorithms are not able to find an optimal solution for this class of problems. In particular, it has been shown that Ant-Colony optimization algorithms suffer from first order deception in the same way as Genetic Algorithms suffer from deception, and second order deception caused by a bias that leads to performance drawbacks over time.
Therefore, in order to implement efficient and effective $ACO$ approaches, the presence of first and second order deception has to be taken into account.

### 3.4.2 Ant-CSP: an Ant-Colony Optimization Algorithm for the CSP

In this section, we provide a detailed description of our $ACO$ approach for the *CSP* problem, called Ant-CSP [FP10].

Given an input set $S$ of $n$ strings of length $m$, over a finite alphabet $\Sigma$, at each iteration of the Ant-CSP algorithm, a *colony* consisting of $u$ ants is generated. Each of the artificial ants, say $colony_i$, searches for a solution by means of the $find\_solution$ procedure, by building a string while it moves

character by character on a table $\tau$, represented as a $|\Sigma| \times m$ matrix. The location $\tau[i, j]$, with $1 \leq i \leq |\Sigma|$ and $0 \leq j \leq m-1$, mantains the pheromone trail for the $i$-th character at the $j$-th position of the string.

Ants choose "their way" probabilistically, using a probability depending on the value $\tau[i, j]$ of the local pheromone trail: the normalized probability for each character is computed, depending on the pheromone value deposited on it. So, the algorithm probabilistically chooses a character.
Initially, $\tau[i, j] = 1/|\Sigma|$; once each ant has built and evaluated its own solution, respectively by means of the $find\_solution()$ and $evaluate\_solution()$ procedures, pheromone trails are updated. We adopted an elitist strategy, so that only the ant that has found the best solution, say $colony_{best}$, updates the pheromone trails, by depositing an amount of pheromone, proportional to the quality of the solution itself, on the characters that appear in the best solution. In particular:

$$\tau^{(t+1)}[i, j] = \tau^{(t)}[i, j] + \left(1 - \frac{HD}{m}\right),$$

where $HD$ is the maximum Hamming distance of the current string from all strings in $S$. Thus, the larger is the pheromone trail for the $i$-th character, the higher will be the probability that this character will be chosen in the next iteration.
Pheromone values are normalized and are used as probabilities. After additional pheromone trail on the best string has been released, the evaporation procedure is applied: this consists in decrementing each value $\tau[i, j]$ by a constant factor $\rho$; in our experiments, we put $\rho = 0.03$.
The pseudo-code of the ANT-CSP algorithm is shown in Algorithm 1.

#### 3.4.2.1 Datasets and Experimental Protocol

We have tested the $ACO$-CSP algorithm using the azotated compounds alphabet $\Sigma = \{A, C, G, T\}$ of the fundamental components of nucleic acids.
In our test platform, we considered a number of input strings $n = 10, 20, 30, 40, 50$, and string length $m = 10, 20, ..., 50, 100, 200, ..., 1000$.

Specifically, we have compared our approach with two state-of-the-art heuristic methods for the $CSP$: a Genetic Algorithm, GA-CSP, and a Simulated Annealing, SA-CSP [LHS05].

**Algorithm 1** Pseudocode of the ANT-CSP algorithm

```
1: initialize table τ
2: for i ← 1 to m do
3:     for j ← 1 to |Σ| do
4:         τ_ij ← 1/|Σ|
5:     end for
6: end for
7: initialize colony
8: while ¬ stopping criterion do
9:     for i ← 1 to u do
10:        colony_i ← new_ant()
11:        colony_i.find_solution()
12:        colony_i.evaluate_solution()
13:    end for
14:    for i ← 1 to m do                    ▷ start pheromone evaporation
15:        for j ← 1 to |Σ| do
16:            τ_ij ← (1 − ρ) · τ_ij
17:        end for
18:    end for                              ▷ end pheromone evaporation
19:    colony_best.update_trails()
20: end while
```

For each of a randomly generated problem instance, all algorithms have been run 20 times. The total colony size for the ANT-CSP algorithm as well as the population size for the GA-CSP algorithm have been set to 10, whereas the number of generations has been set to 1500. In the case of the SA-CSP algorithm, we fixed the number of function evaluations in 15,000, making the number of function evaluations comparable to the computational work performed by the former two algorithms.

Our tests have been performed on an Intel Pentium M 750, 1.86 GHz, 1 GB RAM, running Ubuntu Linux.

#### 3.4.2.2 Experimental Results

We report the results of our tests in the five tables below: *HD* indicates the Hamming distance value, that we aim to minimize, *Time* is the running time in milliseconds. For each length, we computed the average ($AVG$) of the closest string scores found in the 20 runs and the standard deviation $\sigma$. Also, we computed the average of the running time over the 20 runs ($AVG$). The best results are reported in bold.

Experimental results show that almost always the ANT-CSP outperforms both the GA-CSP and the SA-CSP algorithms both in terms of solution quality and efficiency. As a matter of fact, the tables below show that our algorithm is much faster than both the GA-CSP and SA-CSP algorithms. In particular, in the case of short instances, i.e. for $10 \leq m \leq 50$, the ANT-CSP algorithm is from 5 to 36 times faster than GA-CSP.
Furthermore, it turns out that as $n$ increases, the gap between the running time of the ANT-CSP and the SA-CSP algorithms becomes considerable. In Figure 3.1 we report the running times of the algorithms for some sets of strings.

We also remark that the ANT-CSP provides results of a better quality than the other two algorithms in terms of Hamming distance. In fact, the cooperation among the colony ants and the pheromone trails tend to orient the search towards optimal solutions, allowing to explore and modify local optima. On the other hand, the SA-CSP algorithm behaves as a random search, as it simply modifies a string without considering local promising solutions. Likewise, the GA-CSP algorithm performs random mutations and crossovers, whereas the ANT-CSP probabilistically selects each character for the solution.

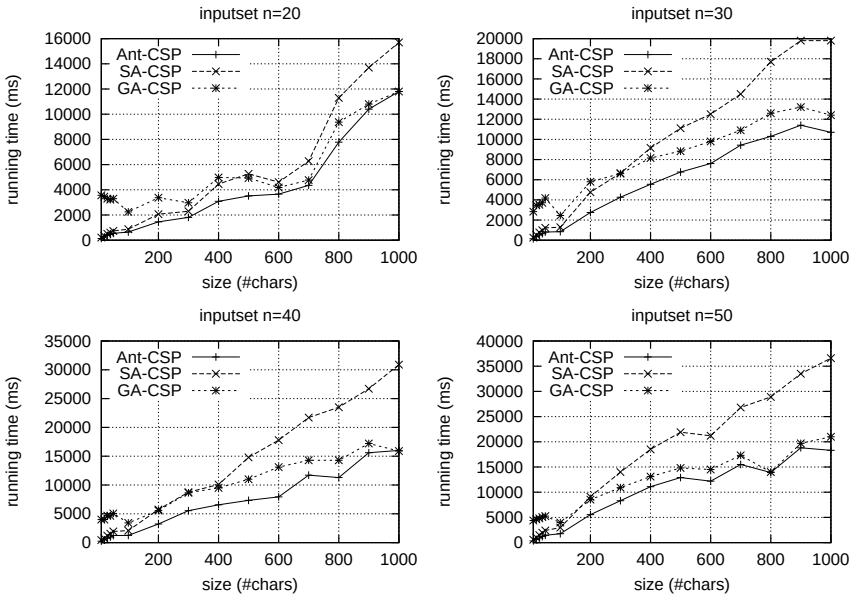


Figure 3.1: Running times plot for $n = 20, 30, 40, 50$.

We note also that the ANT-CSP algorithm is quite robust, as its standard deviation $\sigma$ remains low.

The above considerations show that our algorithm represents a valid and innovative alternative to the SA-CSP and GA-CSP algorithms.

| Size ($m$) | SA-CSP | | | GA-CSP | | | ANT-CSP | | |
|---|---|---|---|---|---|---|---|---|---|
| | HD | | Time | HD | | Time | HD | | Time |
| | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG |
| 10 | 8.45 | 0.497 | 67.5 | **6.9** | 0.3 | 1840 | 7.05 | 0.218 | 50.5 |
| 20 | 15.9 | 0.384 | 112 | 13.3 | 0.714 | 1860 | **13.1** | 0.589 | 97 |
| 30 | 23.6 | 0.663 | 216 | 19.6 | 0.583 | 2700 | **19.3** | 0.557 | 200 |
| 40 | 31.4 | 0.589 | 313 | 25.3 | 0.714 | 3040 | **25.1** | 0.654 | 281 |
| 50 | 38.8 | 0.678 | 428 | 31.8 | 0.994 | 3220 | **31.6** | 0.805 | 386 |
| 100 | 75.9 | 0.943 | 465 | 63.4 | 1.31 | 2060 | **62.2** | 0.766 | 433 |
| 200 | 151 | 1.04 | 901 | 129 | 1.43 | 2290 | **124** | 1.58 | 855 |
| 300 | 226 | 1.18 | 1350 | 195 | 2.19 | 2540 | **188** | 1.57 | 1290 |
| 400 | 301 | 2.01 | 1780 | 262 | 2.52 | 2720 | **252** | 1.68 | 1700 |
| 500 | 375 | 2.05 | 2190 | 330 | 2.52 | 2940 | **317** | 2.15 | 2110 |
| 600 | 450 | 1.87 | 2740 | 400 | 3.71 | 3800 | **385** | 2.5 | 2920 |
| 700 | 525 | 1.68 | 3980 | 470 | 3.43 | 4860 | **451** | 2.95 | 4270 |
| 800 | 600 | 1.51 | 3720 | 540 | 4.04 | 4370 | **517** | 2.11 | 3860 |
| 900 | 675 | 1.19 | 5670 | 610 | 4.01 | 6110 | **585** | 4.05 | 5690 |
| 1000 | 750 | 1.53 | 7720 | 680 | 4.12 | 7850 | **652** | 3.72 | 7850 |

Table 3.1: Results for inputsets of 10 strings of length $m$.

| Size ($m$) | SA-CSP | | | GA-CSP | | | Ant-CSP | | |
|---|---|---|---|---|---|---|---|---|---|
| | HD | | Time | HD | | Time | HD | | Time |
| | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG |
| 10 | 8.95 | 0.384 | 211 | **7.95** | 0.218 | 3560 | **7.95** | 0.218 | 132 |
| 20 | 17.1 | 0.589 | 342 | **14.8** | 0.4 | 3460 | **14.8** | 0.4 | 258 |
| 30 | 24.8 | 0.536 | 502 | 21.6 | 0.497 | 3300 | **21.4** | 0.49 | 370 |
| 40 | 32.5 | 0.497 | 602 | 28.1 | 0.477 | 3220 | **28** | 0.632 | 452 |
| 50 | 40.1 | 0.726 | 735 | 35 | 0.589 | 3300 | **34.8** | 0.536 | 546 |
| 100 | 78.4 | 0.663 | 874 | 69.5 | 0.921 | 2250 | **67.7** | 0.853 | 646 |
| 200 | 154 | 0.917 | 2070 | 140 | 1.74 | 3370 | **135** | 0.963 | 1460 |
| 300 | 229 | 1.16 | 2300 | 210 | 2.09 | 2970 | **203** | 1.95 | 1810 |
| 400 | 305 | 1.18 | 4460 | 281 | 1.95 | 4980 | **272** | 1.56 | 3090 |
| 500 | 380 | 1.25 | 5270 | 353 | 2.52 | 4930 | **341** | 1.65 | 3510 |
| 600 | 456 | 1.46 | 4610 | 426 | 1.89 | 4180 | **411** | 1.68 | 3660 |
| 700 | 531 | 1.16 | 6280 | 499 | 3.51 | 4770 | **482** | 1.95 | 4350 |
| 800 | 607 | 1.32 | 11300 | 572 | 1.88 | 9370 | **553** | 2.84 | 7780 |
| 900 | 682 | 1.49 | 13700 | 645 | 2.58 | 10800 | **623** | 2.51 | 10400 |
| 1000 | 757 | 1.69 | 15700 | 720 | 2.79 | 11800 | **695** | 2.49 | 11800 |

Table 3.2: Results for inputsets of 20 strings of length $m$.

| Size ($m$) | SA-CSP | | | GA-CSP | | | Ant-CSP | | |
|---|---|---|---|---|---|---|---|---|---|
| | HD | | Time | HD | | Time | HD | | Time |
| | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG |
| 10 | 9 | 0 | 245 | 8.25 | 0.433 | 2830 | **8.15** | 0.357 | 148 |
| 20 | 17.3 | 0.458 | 518 | 15.3 | 0.458 | 3460 | **15.2** | 0.4 | 341 |
| 30 | 25.1 | 0.357 | 772 | 22.7 | 0.458 | 3520 | **22.4** | 0.477 | 508 |
| 40 | 33 | 0.316 | 985 | 29.5 | 0.5 | 3720 | **29.1** | 0.357 | 638 |
| 50 | 40.9 | 0.539 | 1230 | 36.9 | 0.357 | 4180 | **36.1** | 0.436 | 814 |
| 100 | 79.3 | 0.557 | 1280 | 72.2 | 0.726 | 2450 | **70.8** | 0.536 | 850 |
| 200 | 156 | 0.829 | 4760 | 144 | 1.08 | 5800 | **140** | 0.975 | 2750 |
| 300 | 232 | 0.831 | 6640 | 216 | 1.77 | 6610 | **209** | 1.27 | 4260 |
| 400 | 308 | 0.829 | 9160 | 290 | 2.93 | 8160 | **280** | 1.28 | 5550 |
| 500 | 383 | 0.963 | 11110 | 362 | 1.66 | 8830 | **351** | 1.79 | 6760 |
| 600 | 459 | 1.24 | 12500 | 436 | 2.14 | 9800 | **423** | 1.95 | 7610 |
| 700 | 534 | 1.03 | 14500 | 510 | 2.57 | 10900 | **495** | 2.01 | 9430 |
| 800 | 610 | 1.14 | 17700 | 583 | 2.57 | 12600 | **568** | 2.36 | 10300 |
| 900 | 686 | 1.69 | 19800 | 658 | 3.42 | 13200 | **640** | 2.09 | 11400 |
| 1000 | 760 | 2.24 | 19800 | 731 | 2.97 | 12400 | **713** | 2.29 | 10700 |

Table 3.3: Results for inputsets of 30 strings of length $m$.

| Size ($m$) | SA-CSP | | | GA-CSP | | | Ant-CSP | | |
|---|---|---|---|---|---|---|---|---|---|
| | HD | | Time | HD | | Time | HD | | Time |
| | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG |
| 10 | 9.4 | 0.49 | 428 | 8.9 | 0.3 | 4000 | **8.55** | 0.497 | 252 |
| 20 | 17.6 | 0.477 | 742 | 15.9 | 0.218 | 3990 | **15.8** | 0.433 | 471 |
| 30 | 25.6 | 0.49 | 1210 | 23.1 | 0.384 | 4690 | **22.9** | 0.384 | 754 |
| 40 | 33.3 | 0.458 | 1540 | 30.4 | 0.572 | 4640 | **30.1** | 0.218 | 962 |
| 50 | 41.2 | 0.433 | 1940 | 37.5 | 0.497 | 5070 | **37** | 0.589 | 1220 |
| 100 | 80 | 0.669 | 2080 | 73.6 | 0.663 | 3420 | **71.7** | 0.477 | 1260 |
| 200 | 157 | 0.889 | 5740 | 146 | 1.24 | 5570 | **142** | 0.669 | 3230 |
| 300 | 233 | 0.889 | 8760 | 219 | 0.954 | 8640 | **214** | 1.05 | 5550 |
| 400 | 309 | 0.831 | 10090 | 293 | 1.87 | 9510 | **285** | 1.16 | 6560 |
| 500 | 385 | 0.748 | 14800 | 368 | 2.07 | 11000 | **358** | 1.24 | 7330 |
| 600 | 461 | 1.01 | 17800 | 441 | 1.69 | 13100 | **431** | 1.91 | 7940 |
| 700 | 536 | 1.05 | 21700 | 515 | 2.1 | 14300 | **503** | 1.01 | 11700 |
| 800 | 612 | 1.1 | 23500 | 590 | 2.34 | 14300 | **577** | 1.93 | 11300 |
| 900 | 688 | 1.34 | 26700 | 664 | 2.52 | 17200 | **649** | 2.31 | 15600 |
| 1000 | 763 | 1.43 | 30900 | 738 | 2.62 | 15900 | **722** | 1.91 | 16000 |

Table 3.4: Results for inputsets of 40 strings of length $m$.

| Size ($m$) | SA-CSP | | | GA-CSP | | | Ant-CSP | | |
|---|---|---|---|---|---|---|---|---|---|
| | HD | | Time | HD | | Time | HD | | Time |
| | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG | AVG | $\sigma$ | AVG |
| 10 | 9.45 | 0.497 | 574 | 9 | 0 | 4390 | **8.85** | 0.357 | 334 |
| 20 | 17.8 | 0.433 | 1030 | 16.2 | 0.4 | 4620 | **16.1** | 0.218 | 620 |
| 30 | 25.9 | 0.3 | 1490 | 23.5 | 0.5 | 4820 | **23.2** | 0.4 | 899 |
| 40 | 33.5 | 0.497 | 1960 | 30.9 | 0.357 | 5070 | **30.6** | 0.497 | 1180 |
| 50 | 41.7 | 0.458 | 2410 | 38.2 | 0.433 | 5270 | **37.8** | 0.433 | 1450 |
| 100 | 80.6 | 0.49 | 2970 | 74.7 | 0.64 | 3970 | **73.3** | 0.64 | 1750 |
| 200 | 158 | 0.671 | 9090 | 148 | 0.91 | 8530 | **144** | 0.698 | 5550 |
| 300 | 234 | 0.678 | 14000 | 222 | 0.91 | 10900 | **216** | 0.889 | 8320 |
| 400 | 310 | 0.792 | 18500 | 297 | 1.65 | 13100 | **289** | 1.41 | 11100 |
| 500 | 386 | 1.16 | 21900 | 369 | 1.69 | 14800 | **362** | 1.24 | 12900 |
| 600 | 462 | 1.13 | 21200 | 444 | 1.5 | 14500 | **434** | 1.74 | 12200 |
| 700 | 538 | 1.14 | 26800 | 519 | 1.9 | 17300 | **508** | 1.7 | 15500 |
| 800 | 614 | 1.43 | 28900 | 594 | 2.9 | 14000 | **582** | 2.29 | 13900 |
| 900 | 689 | 1.1 | 33500 | 667 | 1.64 | 19700 | **656** | 2.11 | 18800 |
| 1000 | 765 | 1.19 | 36600 | 742 | 3.09 | 21000 | **729** | 1.68 | 18300 |

Table 3.5: Results for inputsets of 50 strings of length $m$.

# 3.5 Greedy-Walk and Simulated Annealing for the Closest String Problem

## 3.5.1 An Overview of Greedy-Walk and Simulated Annealing

### 3.5.1.1 The Greedy-walk

In order to provide good upper bounds for the CSP, several heuristics have been proposed in literature. Most of them are characterized by the construction of a good initial solution which allows to improve the running time of branch-and-bound algorithms or other approaches, such as Genetic, Ant-Colony Optimization, or Simulated Annealing algorithms. A straightforward approach for generating a "starting point" in the search space consists in choosing random characters at each position [LHS05, FP10]. Nevertheless, this strategy can lead to a poor choice for a position $j$, $1 \leq j \leq m$, when, for instance, the chosen character does not appear in any of the strings in $S$ at the given position. In this case, other choices would reduce the Hamming distance of the candidate string to some of the strings in $S$.
Another simple, but more effective, heuristic selects the most occurrent character at each position of the initial candidate solution. This idea is adopted, on different levels, in several papers [LHHW08, Jul09, BM10].

In this section, we propose a novel heuristic strategy for building a good initial solution, that combines the most occurrent character approach with a *greedy walk* on the solution space [PCP11]. The underlying idea is to choose, for each position $1 \leq j \leq m$ of the candidate string, the character that along with the previous one minimizes the maximum Hamming distance. We refer to our strategy as *greedy* because the choice at each position is local and blind, as we simply choose a character according to a local evaluation of the distance; moreover, our heuristic identifies a *walk*, since the choice of the character at position $j$ affects the choice of the character at position $j+1$.
In details, given a set $S = \{s_1, s_2, ..., s_n\}$ of strings of the same length $m$, which we represent as an $n \times m$ matrix, we iteratively create a matrix $H$ of the Hamming distance $H_{ij}$, between the candidate string $t$ and the substring $(s_i[1]...s_i[j])$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. For the first column, that involves only the characters appearing in the first po-

| index | ↓ | | | | |
|---|---|---|---|---|---|
| 1 | **A** | G | T | C | T |
| 2 | **A** | T | A | A | A |
| 3 | T | C | G | T | G |
| 4 | **A** | G | T | G | C |
| 5 | G | A | G | A | A |

(a)

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | - | - | - | - |
| 2 | 0 | - | - | - | - |
| 3 | 1 | - | - | - | - |
| 4 | 0 | - | - | - | - |
| 5 | 1 | - | - | - | - |

(b)

Figure 3.2: (a) Input set S. (b) HD matrix after the choice of the most occurrent character.

sition of the strings in $S$, we simply choose the most frequent character, breaking ties arbitrarily; then we compute the Hamming distance between the chosen character and each of the characters $s_1[1], s_2[1], ..., s_n[1]$. To better explain our strategy, we refer to the following set of strings $S = \{AGTCT, ATAAA, TCGTG, AGTGC, GAGAA\}$, reported in Fig. 3.2a. Since the most occurrent character at position 1 of the strings in $S$ is $A$, the first column of the matrix $H$ is initialized as in Fig. 3.2b.

Next, let us consider the case in which $1 < j \leq m$: after locating the rows that led to the maximum Hamming distance at position $j-1$, we choose, for position $j$, any of the characters appearing on these rows (and column $j$); for instance, at position 2 in our example, we can choose between the characters $C$ and $A$, which occur as characters at position $j = 2$ of the strings at rows $i = 3$ and $i = 5$, that have the maximum Hamming distance at position 1. The idea behind this approach is to keep the maximum Hamming distance as low as possible, and this is obtained by trying to not increase its maximum value at each iteration. Suppose we choose character $C$ at position 2; then we compute the Hamming distances between $AC$ and each of the prefixes of length 2 of our strings and move to the position 3, see Fig. 3.3.

We iterate the process until we reach the last position of the strings. In the case where more characters lead to the same Hamming distance, we choose the most occurrent one, breaking ties arbitrarily. In Fig. 3.4a, we report in bold face the character chosen for the solution; the final matrix of the Hamming distances for our example is shown in Fig. 3.4b. It turns out that our proposed heuristic performs better than simply choosing the most occur-

| index | | ↓ | | | |
|---|---|---|---|---|---|
| 1 | A | G | T | C | T |
| 2 | A | T | A | A | A |
| 3 | T | **C** | G | T | G |
| 4 | A | G | T | G | C |
| 5 | G | A | G | A | A |

(a)

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | - | - | - |
| 2 | 0 | 1 | - | - | - |
| 3 | 1 | 1 | - | - | - |
| 4 | 0 | 1 | - | - | - |
| 5 | 1 | **2** | - | - | - |

(b)

Figure 3.3: (a) Input set S. (b) HD matrix after the choice of the most occurrent character.

| index | | | | | |
|---|---|---|---|---|---|
| 1 | **A** | G | T | C | **T** |
| 2 | **A** | T | A | **A** | A |
| 3 | T | **C** | **G** | T | G |
| 4 | **A** | G | T | G | C |
| 5 | G | A | **G** | **A** | A |

(a)

| i \ j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 3 |
| 2 | 0 | 1 | 2 | 2 | 3 |
| 3 | 1 | 1 | 1 | 2 | 3 |
| 4 | 0 | 1 | 2 | 3 | 4 |
| 5 | 1 | 2 | 2 | 2 | 3 |

(b)

Figure 3.4: (a) Input set S. (b) HD matrix after the choice of the most occurrent character.

rent character at each position, as result from extensive experimentations on large datasets.

#### 3.5.1.2 The Simulated Annealing Algorithm

Simulated Annealing (SA, for short) is a generalization of Monte Carlo methods, originally proposed by Metropolis and Ulam [MU49, MRR$^+$53] as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. The basic idea of SA was taken from an analogy with the annealing process used in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. SA is a stochastic method broadly applied to solve continuous and discrete optimization problems: find the global minimum of a cost function with many degrees of freedom is a difficult task, especially when the function is characterized by the presence of many local minima. Methods based on SA apply a probabilistic mechanism to escape from such local minima: the underlying idea is to accept, under certain conditions, not only transitions that improve the objective function value, but also transitions that do not. The probability of accepting worsening steps varies during the search phase, and it slowly decreases to zero. At the end of the search phase, when only improvement of the objective function are considered, this method behaves as a local search. Nevertheless, the possibility of explore points of the search space that deteriorate the current optimal solution, allows to escape local minima and to better explore the set of feasible solutions.

The SA strategy has been inspired by the thermodynamic annealing process of solid materials, such as glass and metals, where a solid is heated to a liquid state and then cooled slowly, so that its structure becomes frozen at the crystal configuration of lowest energy. In this process, temperature plays a major role: when it is set to high values, the atoms in the system are in a highly disordered state, and the energy of the system is high. To reach a more orderly state, the temperature is slowly decreased. The annealing accomplishes this phenomenon by gradually cooling the system, in such a way that it reaches a stable structure. A fast decrement in temperature, in fact, might cause defects in crystal structure. The system is in thermodynamic equilibrium at temperature $T$ if the probability $P(E_i)$ of a state

having energy $E_i$ is governed by the Boltzmann's distribution

$$P(E_i) = \frac{\exp\left(-\frac{E_i}{k_b T}\right)}{\sum_j \exp\left(-\frac{E_j}{k_b T}\right)},$$

where $k_b$ represents the Boltzmann's constant.
In the original Metropolis scheme, an initial state of a thermodynamic system is chosen, having energy $E$ and temperature $T$. Keeping $T$ constant, the initial configuration is perturbed, and the energy change $\Delta E$ is computed. If $\Delta E$ is negative, that happens when the new configuration represents a better point, the new configuration is always accepted, otherwise it is accepted with a probability given by the Boltzmann factor $e^{-(\Delta E/T)}$. This process is repeated $L$ times for the current temperature, then the temperature is decremented and the entire process is repeated until a frozen state is reached at $T = 0$. As already discussed, methods based on the SA may accept not only transitions that lead to better solutions, but also transitions that lead to worse ones, in order to escape local optima. In particular, at the beginning of the search, when temperatures are higher, the algorithm behaves as a random search, therefore bad solutions can be accepted; whereas for lower values of $T$, solutions are located in promising regions of the search space.

#### 3.5.1.3 Simulated Annealing Convergence

The Simulated Annealing method is based on the theory of Markov processes [MTG93], that define models where the successor of the current state is chosen stochastically according only to its predecessor, without considering the entire previous history. In particular, the convergence results for SA have taken into account two models: the first considers a sequence of homogeneous Markov chains, the other a single inhomogeneous Markov chain [HJJ03].
To introduce convergence results for both the homogeneous and inhomogeneous model, some additional notations are needed [BBM08].
Let us define the exponential acceptance rule as follows:

$$Y \leftarrow neighbor(N(X^{(t)}))$$

$$X^{(t+1)} = \begin{cases} Y, & f(Y) \leq f(X^{(t)}) \\ Y, & f(Y) > f(X^{(t)}), \text{with } p = e^{-(f(Y)-f(X^{(t)}))/T} \\ X^{(t)}, & f(Y) > f(X^{(t)}), \text{with } (1-p) \end{cases} \tag{3.7}$$

where $X^{(t+1)}$ represents the configuration at time $t+1$, obtained from the previous state $X^{(t)}$; $p$ is the probability of choosing a worsening solution; $T$ is the temperature parameter.

Let $(\mathscr{X}, f)$ be an instance of a combinatorial optimization problem, where $\mathscr{X}$ represents the search space and $f$ the objective function.
Let $\mathscr{X}^*$ be the set of optimal solutions.
Starting from an initial configuration $X^{(0)}$ and iteratively applying the exponential acceptance rule (3.7), a trajectory $X^{(t)}$ is defined.
Under certain conditions, the probability of finding one of the optimal solutions tends to one when the number of iterations goes to infinity:

$$\lim_{k\to\infty} Pr(X^{(k)} \in \mathscr{X}^*) = 1. \tag{3.8}$$

Let $\mathscr{O}$ denote the set of possible outcomes of a sampling process, and let $X^{(k)}$ be the stochastic variable denoting the outcome of the $k$-th trial. Given the probability that the configuration is at a specific state $j$, after being at the previous state $i$, we define the elements of the transition probability matrix $P$ as follows:

$$p_{ij}(k) = Pr(X^{(k)} = j | X^{(k-1)} = i). \tag{3.9}$$

A stationary distribution of a finite time-homogeneous Markov chain is defined as the stochastic vector $q$, where the components $q_i$ are

$$q_i = \lim_{k\to\infty} Pr(X^{(k)} = i | X^{(0)} = j) \quad \text{for all } j \in \mathscr{O}. \tag{3.10}$$

After the introductory definitions given above, some considerations on the homogeneous and inhomogeneous models can be addressed.
In the homogeneous model, the temperature $T$ is assumed to be held constant for a sufficiently large number of iterations. Therefore, the Markov chain associated to SA has a stationary distribution $q(T)$ whose components are given by:

$$q_i(T) = \frac{e^{-f(i)/T}}{\sum_{j\in\mathscr{X}} e^{-f(j)/T}} \tag{3.11}$$

and

$$\lim_{T\to 0} q_i(T) = q_i^* = \frac{1}{|\mathscr{X}^*|} I_{\mathscr{X}^*}(i) \tag{3.12}$$

where $I_{\mathscr{X}^*}$ is the characteristic function of the set $\mathscr{X}^*$, and it is equal to one if the argument belongs to the set, zero otherwise.
It follows that

$$\lim_{T\to 0} \lim_{k\to\infty} Pr(X^{(k)} \in \mathscr{X}^*) = 1 \tag{3.13}$$

Therefore we can say that the algorithm asymptotically converges with probability one, i.e. it finds an optimal solution with probability one.

The second convergence approach for Simulated Annealing is based on inhomogeneous Markov chain models. In this approach, the Markov chain does not need to reach a stationary distribution, but the condition that the temperature parameter cools down sufficiently slowly before converging is imposed.
At each iteration $k$, a different temperature value $T_k$ is defined; this leads to a nonincreasing sequence of values $T_k$ such that $\lim_{k\to\infty} T_k = 0$.
If the temperature decrement is sufficiently slow, we have

$$T_k \geq \frac{A}{\log(k + k_0)} \tag{3.14}$$

for $A > 0$ and $k_0 > 2$; therefore

$$\lim_{k\to\infty} Pr(X^{(k)} \in \mathscr{X}^*) = 1 \tag{3.15}$$

that means the Markov chain converges in distribution to $q^*$.

The homogeneous Markov chain model has been adopted in [LM86], which proves that the algorithm converges with probability close to one, but there are also cases where convergence takes exponential time. An important result of this proof is the conjecture that when the temperature is close to zero, the second largest eigenvalue will be close to one for problems with local optima, and then the convergence will be very slow [HJJ03]. Therefore, the initial temperature should be set to high values.
An important result of the inhomogeneous approach is the following: while the logarithmic cooling schedule in (3.14) is a sufficient condition for the convergence, for few values the logarithmic rule is also necessary [RSV91,

HJJ03]; moreover, there is a unique choice for which the logarithmic rule is both necessary and sufficient for the convergence of the Simulated Annealing [Haj88].

### 3.5.2 A Combined Greedy-Walk Heuristic and Simulated Annealing Approach for the CSP

Kirkpatrick first proposed to apply SA to solve combinatorial optimization problems [KGV83]. He replaced the notion of energy with a cost function, and the states of a system of particles with the solutions of a minimization problem; thus, the search of a minimum energy status is translated into the search of a solution that minimizes the cost function. Liu *et al.* [LHS05] proposed a SA approach for the CSP that works much along the same lines as Kirkpatrick's algorithm; in a later work [LHHW08], SA has been combined with a GA.

The main innovative point in our SA approach for the *CSP* problem lies in the choice of the initial string and the mutation mechanism implemented [PCP11]. Our algorithm starts the solution search from a string built by applying the *greedy-walk* heuristic proposed above (see Section 3.5.1.1). Unlike the approach proposed in [LHS05], where the initial string is created randomly, our heuristic allows to locate a good starting point, that is a solution in general not too far from the optimal. This strategy not only speeds up the convergence of the algorithm, but finds a better basin of attraction than the ones identified by the "most occurrent character" heuristic.

After setting the initial temperature, the algorithm starts its search. For each temperature value $T$, a block of $L$ iterations is performed. At each iteration, a new string $t'$ of length $m$ over the alphabet $\Sigma$ is constructed by perturbing the string $t$ built during the previous iteration. The idea behind our rearrangement mechanism is to modify the string $t$ at a certain number of random positions, by initially substituting them with the most frequent characters. The number of characters to change is determined by the quality of the solution: the better is the solution, i.e. the lower is the maximum Hamming distance, the smaller is the number of positions which are changed. It should be noted that this step leads to a different solution from the one built by the "most occurrent character" heuristic, since in our

case only few positions of $t$ are modified. In some cases, when this step fails since the positions that we want to change in $t$ already contain the most frequent characters, we try to move to a new search domain, in order to explore new areas of the search space, thus escaping local optima. This is done by substituting some characters with those that appear in the farthest string from $t$ in $S$. This strategy has been already successfully adopted in [MLO+04, GMPV08, LLHM11].
Once a new string $t'$ is obtained from $t$, the energy change $\Delta E = d_H(t', S) - d_H(t, S)$ is evaluated, where $S$ is the input set of strings and $d_H(t', S)$ stands for the maximum Hamming distance between $t$ and the strings in $S$.
If $\Delta E \leq 0$, $t'$ becomes the new current solution, otherwise $t'$ is chosen as current solution with probability $e^{-(\Delta E/T)}$ only. At the end of each block of iterations, the temperature value is multiplied by a reduction factor $\gamma$. The algorithm stops when a suitable termination criterion is met.
The pseudo-code of the algorithm SA for the CSP problem (SAGW) is shown in Algorithm 2.

**Algorithm 2** Pseudocode of the SAGW algorithm

```
1: generate an initial string t by applying the greedy-walk heuristic
2: set an initial T = T_max
3: set the number of iterations L
4: set the reduction factor γ
5: while ¬ stopping criterion do
6:     for 0 ≤ I < L do
7:         t' ← mutate(t);
8:         Δ = evaluate_energy(t', S) − evaluate_energy(t, S);
9:         if (Δ ≤ 0 OR (Δ > 0 AND e^{−Δ/T} > random(0, 1))) then
10:            t ← t';
11:        end if
12:    end for
13:    T ← γ · T;
14: end while
```

#### 3.5.2.1 Datasets and Experimental Protocol

We have tested our approach using a large and comprehensive testbed. In particular, our experimentation focuses on two datasets, consisting of simu-

lated and real biological data. The artificial dataset includes 195 instances, that can be grouped into three main classes, according to the alphabet size $|\Sigma|$. The first class, characterized by an alphabet size $|\Sigma| = 2$, contains instances which can find applications in Coding Theory. The second class uses an alphabet of four characters, that represents the azotated compounds alphabet $\Sigma = \{A, C, G, T\}$ of the fundamental components of nucleic acids. Finally, the third class adopts an alphabet consisting of twenty characters, which denote the 20 amino acids.

In our test platform, we considered a number of input strings $n = 10, 15, 20, 25, 30$, and string lengths $m = 100, 200, ..., 900, 1000, 2000, ..., 5000$. The instances used were randomly generated as follows: given $n$, $m$, and the alphabet $\Sigma$, a character was randomly chosen from $\Sigma$ for each position in the resulting string. For each of the simulated problem instances, the algorithm was run 20 times.

The real dataset consists of six instances from the McClure dataset [MVF94], characterized by an alphabet size $|\Sigma| = 20$; these data represent a set of protein sequences frequently used to test string-comparison algorithms.

The parameters required by the Simulated Annealing algorithm have been set as follows: the initial percentage of characters to change was set to 4%, whereas the starting temperature $T$ was set to 5. As stopping criterion we have used the allowed number of iterations, which we have set in our tests to 10. This means that the temperature value was subjected to 10 decrements; for each of them $L = 500$ iterations were performed. Finally, the temperature reduction factor $\gamma$ was set to 0.8. We remark that all these parameters have been determined experimentally.

#### 3.5.2.2 Experimental Results

Tables 3.6-3.21 report the results of our comparisons with the ILP formulation, which is the most effective approach for the CSP problem. The first two columns, $n$ and $m$, are the number of strings in $S$ and their length, respectively. The ILP column contains the optimal closest string score found by solving the integer linear programming formulation for the problem proposed

in Section 3.3. The ILP problem is solved by using the CPLEX version 12.1[2], with default settings.
The columns named SAGW report the best, average, and worst closest string scores found by our algorithm in the 20 runs. The last column shows the *gap* between our best solution and the one found by the exact method; it reports how much our best solution is far from the best (i.e., in the first table, for $n = 10$ and $m = 100$, our solution is 0.75% worse than the solution found by CPLEX).

It results that the solutions found by our approach are very close to the ones obtained by solving the mathematical IP formulation of the CSP with an exact method, showing the effectiveness of our proposed approach. Moreover, exact methods tend to perform quite poorly on large instances of the problem, especially in terms of CPU time, while our algorithm is not so much affected by the complexity of the datasets. By inspecting the experimental results, it turns out that the gap between our approach and the exact method tends to be smaller as the length $m$ increases. For real instances, our algorithm performs very well in the cases of string length equals to 100 and 98 (see Table 3.21).

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 10 | 100 | 37 | 38 | 39.6 | 51 | 0.008 |
| 10 | 200 | 74 | 75 | 78.05 | 92 | 0.009 |
| 10 | 300 | 111 | 112 | 116.3 | 137 | 0.007 |
| 10 | 400 | 148 | 151 | 154.4 | 178 | 0.008 |
| 10 | 500 | 186 | 189 | 191.95 | 217 | 0.005 |
| 10 | 600 | 224 | 226 | 230.9 | 261 | 0.006 |
| 10 | 700 | 260 | 264 | 269.4 | 303 | 0.006 |
| 10 | 800 | 299 | 301 | 307.4 | 343 | 0.005 |
| 10 | 900 | 334 | 338 | 343.9 | 382 | 0.004 |
| 10 | 1000 | 372 | 377 | 382.15 | 425 | 0.004 |
| 10 | 2000 | 747 | 752 | 759.15 | 839 | 0.003 |
| 10 | 3000 | 1123 | 1129 | 1141.4 | 1253 | 0.003 |
| 10 | 4000 | 1493 | 1496 | 1516.5 | 1662 | 0.002 |
| 10 | 5000 | 1871 | 1877 | 1896.25 | 2083 | 0.002 |

Table 3.6: Results for $n = 10$ and $|\Sigma| = 2$.

[2]http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 15 | 100 | 41 | 41 | 42.75 | 53 | 0.010 |
| 15 | 200 | 78 | 80 | 82.95 | 96 | 0.008 |
| 15 | 300 | 118 | 120 | 124.1 | 141 | 0.009 |
| 15 | 400 | 158 | 161 | 163.4 | 186 | 0.007 |
| 15 | 500 | 197 | 200 | 203.7 | 227 | 0.007 |
| 15 | 600 | 237 | 240 | 243.8 | 272 | 0.006 |
| 15 | 700 | 275 | 277 | 284.35 | 317 | 0.006 |
| 15 | 800 | 316 | 320 | 324.25 | 357 | 0.006 |
| 15 | 900 | 355 | 359 | 362.65 | 402 | 0.005 |
| 15 | 1000 | 393 | 397 | 402.5 | 445 | 0.004 |
| 15 | 2000 | 789 | 796 | 800.05 | 875 | 0.004 |
| 15 | 3000 | 1185 | 1191 | 1198.5 | 1308 | 0.003 |
| 15 | 4000 | 1580 | 1589 | 1597.6 | 1737 | 0.003 |
| 15 | 5000 | 1971 | 1983 | 1994.95 | 2167 | 0.002 |

Table 3.7: Results for $n = 15$ and $|\Sigma| = 2$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 20 | 100 | 42 | 44 | 44.6 | 54 | 0.014 |
| 20 | 200 | 82 | 85 | 87.1 | 103 | 0.014 |
| 20 | 300 | 124 | 128 | 129.25 | 148 | 0.012 |
| 20 | 400 | 163 | 167 | 169.95 | 195 | 0.010 |
| 20 | 500 | 204 | 207 | 212.75 | 242 | 0.010 |
| 20 | 600 | 246 | 250 | 254.85 | 281 | 0.011 |
| 20 | 700 | 285 | 291 | 296.65 | 327 | 0.009 |
| 20 | 800 | 328 | 334 | 337.2 | 369 | 0.009 |
| 20 | 900 | 368 | 374 | 379.45 | 418 | 0.008 |
| 20 | 1000 | 409 | 415 | 420.7 | 465 | 0.008 |
| 20 | 2000 | 819 | 830 | 835.9 | 903 | 0.006 |
| 20 | 3000 | 1231 | 1240 | 1249.55 | 1347 | 0.005 |
| 20 | 4000 | 1636 | 1653 | 1663.25 | 1784 | 0.004 |
| 20 | 5000 | 2054 | 2071 | 2080.4 | 2236 | 0.004 |

Table 3.8: Results for $n = 20$ and $|\Sigma| = 2$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 25 | 100 | 43 | 45 | 46.2 | 58 | 0.017 |
| 25 | 200 | 85 | 87 | 88.95 | 100 | 0.013 |
| 25 | 300 | 126 | 129 | 132.05 | 152 | 0.012 |
| 25 | 400 | 169 | 173 | 175.25 | 194 | 0.011 |
| 25 | 500 | 211 | 215 | 218.05 | 239 | 0.010 |
| 25 | 600 | 252 | 257 | 259.4 | 287 | 0.009 |
| 25 | 700 | 293 | 298 | 302.85 | 334 | 0.008 |
| 25 | 800 | 335 | 340 | 344.8 | 377 | 0.008 |
| 25 | 900 | 377 | 384 | 386.95 | 424 | 0.007 |
| 25 | 1000 | 422 | 427 | 431 | 469 | 0.007 |
| 25 | 2000 | 838 | 848 | 853.7 | 914 | 0.006 |
| 25 | 3000 | 1252 | 1263 | 1275.75 | 1368 | 0.005 |
| 25 | 4000 | 1675 | 1688 | 1698.5 | 1817 | 0.004 |
| 25 | 5000 | 2093 | 2107 | 2117.4 | 2260 | 0.003 |

Table 3.9: Results for $n = 25$ and $|\Sigma| = 2$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 30 | 100 | 45 | 46 | 47.3 | 59 | 0.018 |
| 30 | 200 | 87 | 89 | 91.15 | 105 | 0.015 |
| 30 | 300 | 129 | 132 | 134.6 | 157 | 0.013 |
| 30 | 400 | 173 | 178 | 179.75 | 199 | 0.014 |
| 30 | 500 | 215 | 221 | 222.75 | 249 | 0.012 |
| 30 | 600 | 256 | 263 | 265.8 | 291 | 0.011 |
| 30 | 700 | 299 | 305 | 309.05 | 342 | 0.010 |
| 30 | 800 | 342 | 349 | 352.85 | 382 | 0.010 |
| 30 | 900 | 384 | 391 | 395.85 | 431 | 0.010 |
| 30 | 1000 | 426 | 433 | 439.05 | 481 | 0.009 |
| 30 | 2000 | 853 | 867 | 872.35 | 932 | 0.007 |
| 30 | 3000 | 1277 | 1292 | 1300.85 | 1388 | 0.006 |
| 30 | 4000 | 1707 | 1724 | 1732.05 | 1838 | 0.005 |
| 30 | 5000 | 2133 | 2152 | 2162.25 | 2294 | 0.004 |

Table 3.10: Results for $n = 30$ and $|\Sigma| = 2$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 10 | 100 | 57 | 58 | 60.4 | 78 | 0.015 |
| 10 | 200 | 114 | 117 | 119.05 | 137 | 0.012 |
| 10 | 300 | 172 | 174 | 177.6 | 199 | 0.010 |
| 10 | 400 | 229 | 233 | 236.45 | 260 | 0.009 |
| 10 | 500 | 286 | 290 | 294.2 | 327 | 0.008 |
| 10 | 600 | 346 | 348 | 353.55 | 386 | 0.008 |
| 10 | 700 | 404 | 407 | 412 | 449 | 0.007 |
| 10 | 800 | 462 | 466 | 470.95 | 514 | 0.006 |
| 10 | 900 | 519 | 524 | 528.5 | 576 | 0.006 |
| 10 | 1000 | 578 | 582 | 587.35 | 640 | 0.006 |
| 10 | 2000 | 1154 | 1164 | 1170.55 | 1265 | 0.004 |
| 10 | 3000 | 1731 | 1736 | 1749.85 | 1893 | 0.003 |
| 10 | 4000 | 2314 | 2325 | 2332.45 | 2513 | 0.003 |
| 10 | 5000 | 2891 | 2902 | 2914 | 3142 | 0.003 |

Table 3.11: Results for $n = 10$ and $|\Sigma| = 4$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 15 | 100 | 61 | 63 | 63.85 | 73 | 0.015 |
| 15 | 200 | 122 | 124 | 126.1 | 141 | 0.015 |
| 15 | 300 | 183 | 186 | 188.4 | 211 | 0.013 |
| 15 | 400 | 243 | 246 | 250.45 | 273 | 0.012 |
| 15 | 500 | 304 | 308 | 312.6 | 342 | 0.012 |
| 15 | 600 | 364 | 369 | 373.3 | 406 | 0.009 |
| 15 | 700 | 425 | 430 | 436.05 | 474 | 0.009 |
| 15 | 800 | 487 | 492 | 497.55 | 537 | 0.008 |
| 15 | 900 | 548 | 554 | 558.3 | 606 | 0.008 |
| 15 | 1000 | 609 | 616 | 620.05 | 667 | 0.008 |
| 15 | 2000 | 1219 | 1229 | 1235.05 | 1320 | 0.005 |
| 15 | 3000 | 1828 | 1838 | 1849 | 1975 | 0.004 |
| 15 | 4000 | 2436 | 2452 | 2461.65 | 2634 | 0.004 |
| 15 | 5000 | 3052 | 3071 | 3078.6 | 3293 | 0.003 |

Table 3.12: Results for $n = 15$ and $|\Sigma| = 4$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 20 | 100 | 63 | 65 | 66.8 | 77 | 0.022 |
| 20 | 200 | 125 | 128 | 130.85 | 150 | 0.016 |
| 20 | 300 | 190 | 193 | 195.25 | 215 | 0.015 |
| 20 | 400 | 252 | 257 | 259.5 | 282 | 0.014 |
| 20 | 500 | 313 | 319 | 321.95 | 352 | 0.011 |
| 20 | 600 | 376 | 382 | 385.55 | 418 | 0.010 |
| 20 | 700 | 441 | 448 | 450.1 | 483 | 0.011 |
| 20 | 800 | 503 | 509 | 513 | 554 | 0.009 |
| 20 | 900 | 564 | 572 | 576.95 | 618 | 0.009 |
| 20 | 1000 | 627 | 634 | 639.85 | 691 | 0.009 |
| 20 | 2000 | 1258 | 1268 | 1276.45 | 1359 | 0.007 |
| 20 | 3000 | 1891 | 1902 | 1909.05 | 2028 | 0.005 |
| 20 | 4000 | 2517 | 2531 | 2543.5 | 2697 | 0.004 |
| 20 | 5000 | 3149 | 3164 | 3174.3 | 3365 | 0.004 |

Table 3.13: Results for $n = 20$ and $|\Sigma| = 4$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 25 | 100 | 65 | 67 | 68.45 | 80 | 0.023 |
| 25 | 200 | 128 | 131 | 133.65 | 149 | 0.018 |
| 25 | 300 | 193 | 197 | 199.1 | 217 | 0.015 |
| 25 | 400 | 257 | 262 | 264.95 | 287 | 0.014 |
| 25 | 500 | 321 | 327 | 329.05 | 354 | 0.012 |
| 25 | 600 | 385 | 391 | 395.35 | 423 | 0.013 |
| 25 | 700 | 449 | 455 | 460.1 | 493 | 0.011 |
| 25 | 800 | 514 | 522 | 524.75 | 564 | 0.011 |
| 25 | 900 | 577 | 584 | 589.6 | 629 | 0.010 |
| 25 | 1000 | 644 | 651 | 655.55 | 699 | 0.010 |
| 25 | 2000 | 1283 | 1297 | 1302.25 | 1380 | 0.007 |
| 25 | 3000 | 1928 | 1941 | 1949.05 | 2059 | 0.006 |
| 25 | 4000 | 2571 | 2588 | 2597.65 | 2737 | 0.005 |
| 25 | 5000 | 3213 | 3234 | 3242.6 | 3420 | 0.004 |

Table 3.14: Results for $n = 25$ and $|\Sigma| = 4$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 30 | 100 | 66 | 68 | 69.75 | 83 | 0.024 |
| 30 | 200 | 131 | 134 | 136.9 | 152 | 0.021 |
| 30 | 300 | 197 | 201 | 203.45 | 227 | 0.018 |
| 30 | 400 | 261 | 267 | 269.1 | 295 | 0.015 |
| 30 | 500 | 327 | 332 | 335.25 | 362 | 0.013 |
| 30 | 600 | 392 | 400 | 401.55 | 431 | 0.013 |
| 30 | 700 | 457 | 464 | 468.15 | 499 | 0.012 |
| 30 | 800 | 523 | 529 | 532.85 | 566 | 0.011 |
| 30 | 900 | 588 | 597 | 600.3 | 636 | 0.012 |
| 30 | 1000 | 652 | 659 | 665.35 | 704 | 0.010 |
| 30 | 2000 | 1302 | 1317 | 1323.6 | 1393 | 0.008 |
| 30 | 3000 | 1958 | 1973 | 1979.45 | 2084 | 0.006 |
| 30 | 4000 | 2610 | 2634 | 2637.25 | 2775 | 0.005 |
| 30 | 5000 | 3260 | 3280 | 3291.15 | 3460 | 0.005 |

Table 3.15: Results for $n = 30$ and $|\Sigma| = 4$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 10 | 100 | 78 | 79 | 80.2 | 91 | 0.018 |
| 10 | 200 | 155 | 157 | 159.5 | 178 | 0.014 |
| 10 | 300 | 232 | 235 | 237.75 | 252 | 0.009 |
| 10 | 400 | 310 | 313 | 316.45 | 335 | 0.008 |
| 10 | 500 | 389 | 392 | 396.3 | 420 | 0.009 |
| 10 | 600 | 467 | 469 | 473.5 | 496 | 0.007 |
| 10 | 700 | 546 | 550 | 552.7 | 581 | 0.006 |
| 10 | 800 | 623 | 626 | 631.45 | 665 | 0.006 |
| 10 | 900 | 703 | 706 | 710.55 | 744 | 0.006 |
| 10 | 1000 | 779 | 783 | 787.6 | 829 | 0.006 |
| 10 | 2000 | 1560 | 1564 | 1572.45 | 1643 | 0.004 |
| 10 | 3000 | 2340 | 2347 | 2355.8 | 2452 | 0.003 |
| 10 | 4000 | 3122 | 3131 | 3140 | 3264 | 0.003 |
| 10 | 5000 | 3908 | 3918 | 3924 | 4077 | 0.002 |

Table 3.16: Results for $n = 10$ and $|\Sigma| = 20$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 15 | 100 | 82 | 84 | 84.95 | 93 | 0.024 |
| 15 | 200 | 164 | 165 | 167.35 | 181 | 0.014 |
| 15 | 300 | 245 | 249 | 250.6 | 265 | 0.013 |
| 15 | 400 | 327 | 331 | 333.2 | 348 | 0.012 |
| 15 | 500 | 409 | 413 | 415.6 | 436 | 0.010 |
| 15 | 600 | 490 | 494 | 497.6 | 520 | 0.009 |
| 15 | 700 | 571 | 576 | 580.3 | 605 | 0.009 |
| 15 | 800 | 655 | 660 | 662.7 | 688 | 0.008 |
| 15 | 900 | 735 | 740 | 744.9 | 776 | 0.007 |
| 15 | 1000 | 818 | 822 | 827.5 | 859 | 0.007 |
| 15 | 2000 | 1636 | 1644 | 1650.1 | 1709 | 0.005 |
| 15 | 3000 | 2454 | 2464 | 2472.15 | 2554 | 0.004 |
| 15 | 4000 | 3275 | 3284 | 3292.4 | 3403 | 0.003 |
| 15 | 5000 | 4091 | 4105 | 4115 | 4251 | 0.003 |

Table 3.17: Results for $n = 15$ and $|\Sigma| = 20$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 20 | 100 | 84 | 86 | 87.3 | 94 | 0.027 |
| 20 | 200 | 168 | 170 | 171.55 | 184 | 0.017 |
| 20 | 300 | 250 | 255 | 256.6 | 274 | 0.016 |
| 20 | 400 | 335 | 339 | 340.65 | 359 | 0.012 |
| 20 | 500 | 418 | 422 | 425.25 | 446 | 0.011 |
| 20 | 600 | 502 | 507 | 510.4 | 534 | 0.011 |
| 20 | 700 | 586 | 592 | 594.3 | 618 | 0.010 |
| 20 | 800 | 669 | 674 | 678.35 | 704 | 0.009 |
| 20 | 900 | 754 | 761 | 763.5 | 793 | 0.009 |
| 20 | 1000 | 836 | 845 | 847.05 | 881 | 0.008 |
| 20 | 2000 | 1674 | 1683 | 1688.75 | 1743 | 0.006 |
| 20 | 3000 | 2513 | 2524 | 2530.5 | 2612 | 0.004 |
| 20 | 4000 | 3351 | 3361 | 3372.1 | 3480 | 0.004 |
| 20 | 5000 | 4188 | 4203 | 4203 | 4337 | 0.003 |

Table 3.18: Results for $n = 20$ and $|\Sigma| = 20$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 25 | 100 | 85 | 87 | 88.55 | 94 | 0.027 |
| 25 | 200 | 170 | 173 | 174.65 | 187 | 0.019 |
| 25 | 300 | 255 | 259 | 261.1 | 274 | 0.017 |
| 25 | 400 | 340 | 344 | 346.7 | 364 | 0.013 |
| 25 | 500 | 426 | 430 | 432.95 | 449 | 0.013 |
| 25 | 600 | 510 | 516 | 518.35 | 541 | 0.011 |
| 25 | 700 | 595 | 601 | 604.95 | 628 | 0.012 |
| 25 | 800 | 680 | 687 | 690.1 | 716 | 0.010 |
| 25 | 900 | 766 | 773 | 775.4 | 807 | 0.009 |
| 25 | 1000 | 851 | 859 | 861.2 | 892 | 0.009 |
| 25 | 2000 | 1702 | 1710 | 1716.7 | 1776 | 0.006 |
| 25 | 3000 | 2554 | 2562 | 2570.65 | 2650 | 0.005 |
| 25 | 4000 | 3401 | 3418 | 3424 | 3530 | 0.004 |
| 25 | 5000 | 4255 | 4273 | 4278.8 | 4412 | 0.004 |

Table 3.19: Results for $n = 25$ and $|\Sigma| = 20$.

| | | ILP | SAGW | | | |
|---|---|---|---|---|---|---|
| n | m | Best | Best | Avg | Worst | GAP |
| 30 | 100 | 86 | 89 | 89.65 | 96 | 0.027 |
| 30 | 200 | 173 | 176 | 176.95 | 187 | 0.020 |
| 30 | 300 | 258 | 262 | 264.35 | 278 | 0.017 |
| 30 | 400 | 344 | 349 | 351.05 | 367 | 0.014 |
| 30 | 500 | 430 | 435 | 437.9 | 456 | 0.014 |
| 30 | 600 | 516 | 522 | 525 | 546 | 0.012 |
| 30 | 700 | 602 | 609 | 611.7 | 635 | 0.012 |
| 30 | 800 | 688 | 695 | 698.05 | 721 | 0.010 |
| 30 | 900 | 774 | 782 | 785.1 | 814 | 0.010 |
| 30 | 1000 | 860 | 867 | 871.1 | 901 | 0.010 |
| 30 | 2000 | 1720 | 1732 | 1737.85 | 1788 | 0.007 |
| 30 | 3000 | 2584 | 2594 | 2599.95 | 2677 | 0.005 |
| 30 | 4000 | 3444 | 3460 | 3465 | 3566 | 0.005 |
| 30 | 5000 | 4304 | 4322 | 4328.2 | 4452 | 0.004 |

Table 3.20: Results for $n = 30$ and $|\Sigma| = 20$.

| | | | ILP | SAGW | | | GAP |
|---|---|---|---|---|---|---|---|
| Name | n | m | Best | Best | Avg | Worst | |
| McClure-582-20-6-141 | 6 | 141 | 88 | 95 | 95 | 113 | 0.050 |
| McClure-582-20-10-141 | 10 | 141 | 97 | 106 | 106 | 121 | 0.064 |
| McClure-582-20-12-141 | 12 | 141 | 97 | 105 | 105 | 119 | 0.057 |
| McClure-586-20-6-100 | 6 | 100 | 72 | 73 | 73 | 81 | 0.010 |
| McClure-586-20-10-98 | 10 | 98 | 75 | 77 | 77 | 85 | 0.020 |
| McClure-586-20-12-98 | 12 | 98 | 77 | 79 | 79.05 | 84 | 0.021 |

Table 3.21: Results for McClure instances.

In order to assess the effectiveness of the new heuristic method proposed, we also compared our algorithm with other heuristic algorithms proposed for the *CSP*. It results that our approach performs better than the memetic algorithm (for short, MA) and compounded GA and SA (for short, CGSA) presented in [BM10, LHHW08], respectively. Detailed results are presented in Table 3.22; in boldface we report the best solution and in italic the best average scores. In particular, the columns named CGSA and MA represent, respectively, the best solution found by the compounded GA and SA in one run and the average of the closest string scores found by the memetic algorithm in 10 runs, as reported in their respective papers [BM10, LHHW08], for the alphabet size $|\Sigma| = 2$. Our algorithm SAGW finds *always* better results than the compounded approach, and it also outperforms the average scores of the memetic algorithm in 9 cases out of 18, when $|\Sigma| = 2$.

In Table 3.23, we report the experimental results for the memetic algorithm [BM10], the Genetic Algorithms presented in [Jul09], and our algorithm SAGW on a dataset consisting of 10 strings of length 500, for each alphabet size $|\Sigma| = \{2, 4, 20\}$. In particular, in boldface we report the best solution scores. Since the results in the original papers [BM10, Jul09] are computed for five classes of instances, each of them with a specific Hamming diameter (that is the maximum Hamming distance between any two strings in the input set $S$), we compared the best score among the five classes and the mean of the average values. It turns out that the algorithm SAGW outperforms the Genetic Algorithms when $|\Sigma| = 4$ and $|\Sigma| = 20$, and it finds results that are comparable with the memetic approach.

The algorithm SAGW gives also improved solutions with respect to the Genetic Algorithm and Simulated Annealing algorithm presented in [LHS05] for a small dataset, and implemented in [FP10] for large instances, and it computes always better solutions than the ant-colony optimization approach proposed in [FP10], for $|\Sigma| = 4$, as shown in Table 3.24, where the best average solution scores are reported in italic.

For the simulated data, the best solution found by SAGW always outperforms the ones found by the Largest Distance Decreasing algorithm (for short, LDDA) presented in [LLHM11], as reported in Tables 3.25, 3.26, and 3.27, where we report the best solution in boldface, and the best average

scores in italic. In the case of real instances, our approach guarantees results comparable with LLDA, and outperforms it in two cases (see Table 3.28).

For an in-depth comparison, we refer the reader to [BM10, LHHW08, LLHM11, Jul09, LHS05, FP10].

| | | CGSA | MA | SAGW | |
|---|---|---|---|---|---|
| n | m | Best | Avg | Best | Avg |
| 10 | 300 | 121 | 116.70 | **112** | *116.3* |
| 10 | 400 | 162 | *153.50* | **151** | 154.4 |
| 10 | 500 | 201 | *191.80* | **189** | 191.95 |
| 10 | 600 | 241 | *228.90* | **226** | 230.9 |
| 10 | 700 | 284 | *266.50* | **264** | 269.4 |
| 10 | 800 | 328 | *302.90* | **301** | 307.4 |
| 15 | 300 | 130 | 131.11 | **120** | *124.1* |
| 15 | 400 | 172 | 170.30 | **161** | *163.4* |
| 15 | 500 | 215 | 215.90 | **200** | *203.7* |
| 15 | 600 | 256 | 254.5 | **240** | *243.8* |
| 15 | 700 | 299 | 296.60 | **277** | *284.35* |
| 15 | 800 | 347 | 339.20 | **320** | *324.25* |
| 20 | 300 | 136 | 130.10 | **128** | *129.25* |
| 20 | 400 | 177 | 172.20 | **167** | *169.95* |
| 20 | 500 | 224 | *211.40* | **207** | 212.75 |
| 20 | 600 | 270 | *254.10* | **250** | 254.85 |
| 20 | 700 | 311 | *293.90* | **291** | 296.65 |
| 20 | 800 | 356 | *335.40* | **334** | 337.2 |

Table 3.22: Comparison among MA, CGSA, and our approach for $|\Sigma| = 2$.

| | | | DBGA | | MA | | SAGW | |
|---|---|---|---|---|---|---|---|---|
| $\|\Sigma\|$ | n | m | Best | Avg | Best | Avg | Best | Avg |
| 2 | 10 | 500 | 187 | 191.52 | **186** | 190.44 | 189 | 191.95 |
| 4 | 10 | 500 | 292 | 294.66 | **289** | 291.16 | 290 | 294.4 |
| 20 | 10 | 500 | 394 | 396.54 | **389** | 391.88 | 392 | 396.3 |

Table 3.23: Comparison among the DBGA, MA, and our approach for $n = 10$, $m = 500$.

| | | SA | GA | Ant-CSP | SAGW | |
|---|---|---|---|---|---|---|
| n | m | Avg | Avg | Avg | Best | Avg |
| 10 | 100 | 75.9 | 63.4 | 62.2 | 58 | *60.4* |
| 10 | 200 | 151 | 129 | 124 | 117 | *119.05* |
| 10 | 300 | 226 | 195 | 188 | 174 | *177.6* |
| 10 | 400 | 301 | 262 | 252 | 233 | *236.45* |
| 10 | 500 | 375 | 330 | 317 | 290 | *294.2* |
| 10 | 600 | 450 | 400 | 385 | 348 | *353.55* |
| 10 | 700 | 525 | 470 | 451 | 407 | *412* |
| 10 | 800 | 600 | 540 | 517 | 466 | *470.95* |
| 10 | 900 | 675 | 610 | 585 | 524 | *528.5* |
| 10 | 1000 | 750 | 680 | 652 | 582 | *587.35* |
| 20 | 100 | 78.4 | 69.5 | 67.7 | 65 | *66.8* |
| 20 | 200 | 154 | 140 | 135 | 128 | *130.85* |
| 20 | 300 | 229 | 210 | 203 | 193 | *195.25* |
| 20 | 400 | 305 | 281 | 272 | 257 | *259.5* |
| 20 | 500 | 380 | 353 | 341 | 319 | *321.95* |
| 20 | 600 | 456 | 426 | 411 | 382 | *385.55* |
| 20 | 700 | 531 | 499 | 482 | 448 | *450.1* |
| 20 | 800 | 607 | 572 | 553 | 509 | *513* |
| 20 | 900 | 682 | 645 | 623 | 572 | *576.95* |
| 20 | 1000 | 757 | 720 | 695 | 634 | *639.85* |
| 30 | 100 | 79.3 | 72.2 | 70.8 | 68 | *69.75* |
| 30 | 200 | 156 | 144 | 140 | 134 | *136.9* |
| 30 | 300 | 232 | 216 | 209 | 201 | *203.45* |
| 30 | 400 | 308 | 290 | 280 | 267 | *269.1* |
| 30 | 500 | 383 | 362 | 351 | 332 | *335.25* |
| 30 | 600 | 459 | 436 | 423 | 400 | *401.55* |
| 30 | 700 | 534 | 510 | 495 | 464 | *468.15* |
| 30 | 800 | 610 | 583 | 568 | 529 | *532.85* |
| 30 | 900 | 686 | 658 | 640 | 597 | *600.3* |
| 30 | 1000 | 760 | 731 | 713 | 659 | *665.35* |

Table 3.24: Comparison among SA, GA, Ant-CSP, and our approach for $|\Sigma| = 4$.

| | | LLDA | | | SAGW | | |
|---|---|---|---|---|---|---|---|
| n | m | Best | Avg | Worst | Best | Avg | Worst |
| 10 | 100 | 40 | 40.3 | 41 | **38** | *39.6* | 51 |
| 10 | 200 | 76 | *76.7* | 78 | **75** | 78.05 | 92 |
| 10 | 300 | 116 | 116.7 | 118 | **112** | *116.3* | 137 |
| 10 | 1000 | 379 | *380* | 382 | **377** | 382.15 | 425 |
| 10 | 2000 | 755 | *758* | 760 | **752** | 759.15 | 839 |
| 10 | 3000 | 1131 | *1135.7* | 1141 | **1129** | 1141.4 | 1253 |
| 10 | 4000 | 1516 | 1524 | 1531 | **1496** | *1516.5* | 1662 |
| 10 | 5000 | 1885 | *1888.3* | 1891 | **1877** | 1896.25 | 2083 |
| 20 | 100 | **44** | 44.7 | 45 | **44** | *44.6* | 54 |
| 20 | 200 | 86 | *86.3* | 87 | **85** | 87.1 | 103 |
| 20 | 300 | **127** | *129* | 131 | 128 | 129.25 | 148 |
| 20 | 1000 | 421 | 421.7 | 423 | **415** | *420.7* | 465 |
| 20 | 2000 | 837 | 838.3 | 840 | **830** | *835.9* | 903 |
| 20 | 3000 | 1249 | 1251.3 | 1254 | **1240** | *1249.55* | 1347 |
| 20 | 4000 | 1661 | 1665.7 | 1670 | **1653** | *1663.25* | 1784 |
| 20 | 5000 | 2075 | *2077* | 2081 | **2071** | 2080.4 | 2236 |
| 30 | 1000 | 440 | 441 | 443 | **433** | *439.05* | 481 |
| 30 | 2000 | 870 | *871* | 872 | **867** | 872.35 | 932 |
| 30 | 3000 | 1302 | 1304 | 1305 | **1292** | *1300.85* | 1388 |
| 30 | 4000 | 1725 | *1730.7* | 1733 | **1724** | 1732.05 | 1838 |
| 30 | 5000 | 2159 | *2160.7* | 2163 | **2152** | 2162.25 | 2294 |

Table 3.25: Comparison between LLDA and SAGW for $|\Sigma| = 2$.

| | | LLDA | | | SAGW | | |
|---|---|---|---|---|---|---|---|
| n | m | Best | Avg | Worst | Best | Avg | Worst |
| 10 | 100 | 60 | 60.7 | 61 | **58** | *60.4* | 78 |
| 10 | 200 | 119 | 120 | 121 | **117** | *119.05* | 137 |
| 10 | 300 | 177 | 177.7 | 179 | **174** | *177.6* | 199 |
| 10 | 1000 | 588 | 588.7 | 589 | **582** | *587.35* | 640 |
| 10 | 2000 | 1167 | 1172.3 | 1181 | **1164** | *1170.55* | 1265 |
| 10 | 3000 | 1752 | 1753.3 | 1755 | **1736** | *1749.85* | 1893 |
| 10 | 4000 | 2346 | 2348 | 2349 | **2325** | *2332.45* | 2513 |
| 10 | 5000 | 2908 | 2915.3 | 2921 | **2902** | *2914* | 3142 |
| 20 | 100 | 67 | 67.3 | 68 | **65** | *66.8* | 77 |
| 20 | 200 | 131 | 131.3 | 132 | **128** | *130.85* | 150 |
| 20 | 300 | 196 | 196.3 | 197 | **193** | *195.25* | 215 |
| 20 | 1000 | 644 | 646.7 | 649 | **634** | *639.85* | 691 |
| 20 | 2000 | 1280 | 1282.3 | 1287 | **1268** | *1276.45* | 1359 |
| 20 | 3000 | 1918 | 1922 | 1926 | **1902** | *1909.05* | 2028 |
| 20 | 4000 | 2544 | 2551.7 | 2562 | **2531** | *2543.5* | 2697 |
| 20 | 5000 | 3180 | 3185.3 | 3193 | **3164** | *3174.3* | 3365 |
| 30 | 1000 | 672 | 673.3 | 675 | **659** | *665.35* | 704 |
| 30 | 2000 | 1324 | 1328 | 1332 | **1317** | *1323.6* | 1393 |
| 30 | 3000 | 1984 | 1987.3 | 1992 | **1973** | *1979.45* | 2084 |
| 30 | 4000 | **2628** | *2631.7* | 2637 | 2634 | 2637.5 | 2775 |
| 30 | 5000 | 3296 | 3298.7 | 3301 | **3280** | *3291.15* | 3460 |

Table 3.26: Comparison between LLDA and SAGW for $|\Sigma| = 4$.

| | | LLDA | | | SAGW | | |
|---|---|---|---|---|---|---|---|
| n | m | Best | Avg | Worst | Best | Avg | Worst |
| 10 | 100 | 82 | 83 | 84 | **79** | *80.2* | 91 |
| 10 | 200 | 161 | 161.7 | 163 | **157** | *159.5* | 178 |
| 10 | 300 | 240 | 240 | 240 | **235** | *237.75* | 252 |
| 10 | 1000 | 787 | 790 | 794 | **783** | *787.6* | 829 |
| 10 | 2000 | 1570 | 1573.7 | 1578 | **1564** | *1572.45* | 1643 |
| 10 | 3000 | 2352 | *2353.3* | 2355 | **2347** | 2355.8 | 2452 |
| 10 | 4000 | 3132 | *3138.3* | 3143 | **3131** | 3140 | 3264 |
| 10 | 5000 | **3918** | 3923 | 3927 | **3918** | 3924 | 4077 |
| 20 | 100 | 87 | 87.7 | 88 | **86** | *87.3* | 94 |
| 20 | 200 | 172 | 173.3 | 174 | **170** | *171.55* | 184 |
| 20 | 300 | 256 | 257 | 258 | **255** | *256.6* | 274 |
| 20 | 1000 | 846 | 848 | 851 | **845** | *847.05* | 881 |
| 20 | 2000 | 1689 | 1690.3 | 1692 | **1683** | *1688.75* | 1743 |
| 20 | 3000 | 2531 | 2534 | 2537 | **2524** | *2530.5* | 2612 |
| 20 | 4000 | 3367 | *3369.7* | 3374 | **3361** | 3372.1 | 3480 |
| 20 | 5000 | 4208 | 4212.3 | 4218 | **4203** | *4203* | 4337 |
| 30 | 1000 | 874 | 874.7 | 875 | **867** | *871.1* | 901 |
| 30 | 2000 | 1742 | 1742 | 1742 | **1732** | *1737.85* | 1788 |
| 30 | 3000 | 2607 | 2608.3 | 2610 | **2594** | *2599.95* | 2677 |
| 30 | 4000 | 3472 | 3473.7 | 3475 | **3460** | *3465* | 3566 |
| 30 | 5000 | 4343 | 4344.3 | 4347 | **4322** | *4328.2* | 4452 |

Table 3.27: Comparison between LLDA and SAGW for $|\Sigma| = 20$.

| | | | LLDA | SAGW | | |
|---|---|---|---|---|---|---|
| Name | n | m | Avg | Best | Avg | Worst |
| McClure-582-20-6-141 | 6 | 141 | **89** | 95 | 95 | 113 |
| McClure-582-20-10-141 | 10 | 141 | **100.3** | 106 | 106 | 121 |
| McClure-582-20-12-141 | 12 | 141 | **100.7** | 105 | 105 | 119 |
| McClure-586-20-6-100 | 6 | 100 | **71.3** | 73 | 73 | 81 |
| McClure-586-20-10-98 | 10 | 98 | 79.3 | **77** | 77 | 85 |
| McClure-586-20-12-98 | 12 | 98 | 81 | **79** | 79.05 | 84 |

Table 3.28: Comparisons between SAGW and LLDA [LLHM11] for McClure instances.

## 3.6 Conclusions and Future Directions

Algorithms for sequence analysis are of central importance in computational molecular biology and, more in general, in genetics and proteomics. The recognition of similarities and analogies in biological samples, in fact, finds applications in many fields of biomedicine, such as the design of genetic drug and genetic probes, and in locating binding sites. The *Closest String Problem* belongs to this class of problems, and it informally defines the task of finding a pattern that presents similarities with a given set of sequences.
In this chapter, we introduced first the biological implications of string search problems, and after an accurate analysis of the state-of-the-art methods, we presented two promising approaches for the *CSP*.
In particular, the ANT-CSP, based on the *Ant-Colony Optimization* metaheuristic [FP10], and a Simulated Annealing method combined with a *greedy-walk* heuristic [PCP11], have been introduced. ANT-CSP has been compared with two heuristic approaches presented for the *CSP*, that are a Genetic Algorithm and a Simulated Annealing method [LHS05]. Experimental results show that our algorithm computes almost always better solutions and is much faster than both the two methods, regardless the number and the length of input strings. The *greedy walk* heuristic presented else in this chapter, as the name states, identifies a *walk* on the solution space, *greedy* because the choice of each component of the solution is local and blind. We tested the effectiveness of such heuristic by combining it with a Simulated Annealing algorithm for the *CSP*. We compared our approach with the mathematical formulation proposed in [BDLRP97, LMW02, MLO$^+$04] and the heuristic methods presented in [BM10, LHHW08, LLHM11, Jul09, LHS05, FP10]. Experimental results show that the combination of our heuristic with the SA algorithm computes almost always better solutions than the other heuristic approaches, and has comparable results with exact methods. Despite the large number of methods presented in literature for the *Closest String Problem*, it is our opinion there is still a lot of work to investigate the computational issues of another important problem occurring in string selection, that is the *Closest Substring Problem*. For this reason, we plan to focus our future works on two fronts: on one side, performance improvements and development of a new strategy to improve quality of solutions and convergence speed; on the other hand, we plan to extend our approaches to the *Closest Substring Problem*, due to its biological and computational relevance.

# 4

# Probe Design and Selection Problems

*"Volumes of history written in the ancient alphabet of G and C, A and T."*

Sy Montgomery

## 4.1 Introduction

One of the most important recent development in biology is the success of The Human Genome Project (HGP), whose principal goal was to sequence the entire human genome. The project was completed in 2003, after a 13-years effort; due to the great amount of data collected, the data analysis on this project will continue for years. In this scenario, *in silico* design has become an effective approach to tackle genomic data, since computational methods allow one to reduce the time and cost required for performing experiments, by decreasing the number of wet experiments needed or by speeding up the experimentation process.

The first step towards mapping the human genome consists in breaking down the DNA into segments, achieved by making a genomic library from human DNA. A DNA library is constructed by fragmenting large pieces of DNA with a restriction enzyme, and cloning these fragments randomly into a suitable vector. Given a DNA library, the problem is how to identify a specific probe in a clone. A probe is a segment of DNA or RNA, labelled with a radioactive isotope, dye, or enzyme, used to find a specific sequence of nucleotides or gene on a DNA molecule by hybridization. A clone is said to be positive if it contains a given probe, otherwise it is negative.

DNA microarray technology provides a fast approach to monitor thousands

of genes at the same time on a single chip, and to perform hybridization experiments: the basic mechanism behind microarrays is hybridization between two DNA strands, according to the Watson-Crick complementary pairing of nucleotide bases. A DNA microarray or DNA chip is a glass or nylon slide, onto which single strands of DNA sequences, that represent the probes, are fixed. The foundation of microarray technology lies in the Watson-Crick base-pairing: two DNA strands, the probe and the target, hybridize if they are complementary to each other. The amount of hybridized target represents the gene expression level.
Some important applications of DNA microarray are [SL03]: gene identification, where microarrays are used to measure the specific expression of thousands of genes simultaneously, in order to identify their functions and how they determine phenotypes; detection of genetic disease [AP03]; temporal analysis of cell cycle variation [EB00]; drug discovery and toxicological research [DG99]; forensic identification [HSR+08].

Since the optimal design involves the minimum number of hybridization experiments, the selection of the probe set represents a crucial step in the entire process, because it affects the experimental costs and its effectiveness [BCDV+01]. Problems involving the selection of probe sets to analyze unknown clones fall into the category of Probe Selection Problems, and can be classified into two main classes: *Oligonucleotide Fingerprinting* and *Non-Unique Probe Selection Problem*.

**Oligonucleotide Fingerprinting** Oligonucleotide fingerprinting [HPM+99, HSS+00, HSL+00, GLSW07, POP11a] is a technique used to identify unique probes by performing hybridization experiments: hybridization signals are evaluated and a vector of numerical values, the fingerprint, is assigned to each probe. Essentially, a fingerprint is a sequence of numbers representing the interaction of the clone sequence with the probe sets. The oligonucleotide fingerprinting method was first developed in 1986 for analyzing the mammalian genome. From then on, since many computational challenges arise in this method, several algorithmic approaches have been developed; in particular, two main issues have been handled: clones classification and probe selection. The classification of oligonucleotide fingerprints comprises many steps, starting from the creation of clones to perform hybridization, to the analysis of the fingerprints resulting from the experiments. Once this

analysis is completed, clones are classified according to their characteristics and the results of hybridization; two classes are therefore constructed, representing the clones that hybridize to probes, and those that do not [FBJ03]. Unfortunately, the step of translating hybridization signals into two groups, representing the presence of an hybridization and its absence, is not easy to perform, especially due to the intensity values of hybridization array [FBJ03]. Most of the existing approaches to classify clones are based on clustering methods [HPM$^+$99, HSS$^+$00, HSL$^+$00].
Probe selection for fingerprinting is a difficult task, since the "quality" of probes affects the final design. From this perspective, effective and efficient computational methods have been developed to provide a fast and accurate set of probes. These approaches range from exact methods [RG02], to heuristic algorithms [GLSW07]. In particular, two variations of the original probe selection problem have been introduced by Borneman *et al.* in [BCDV$^+$01], where the objectives are to maximize the number of clones distinguished (*Maximum Distinguishing Probe Set*), and to minimize the number of probes needed to distinguish all the clones (*Minimum Cost Probe Set*). The *MCPS* problem can be considered as a special case of the SET COVER problem [Hoc97], and *MDPS* as a special case of MAXIMUM COVERAGE [Hoc97]. *MCPS* and *MDPS* are NP-hard, when the length of probes is unbounded [BCDV$^+$01]. To overcome their NP-hardness, some heuristic methods have been introduced in literature [BCDV$^+$01, CMN04, dST93].

**Non-Unique Probe Selection Problem** Depending on the application, hybridization experiments can be conducted using unique or non-unique probes: the first case occurs when a probe hybridizes to only one target, otherwise the presence of non-unique probes characterizes the experiment [GLSW07]. Unfortunately, performing hybridization experiments by using unique probes for every target is not always possible, because targets might present similar traits. Moreover, false positive and false negative can invalidate the results of hybridization experiments. The approach adopted to overcome these difficulties consists in using non-unique probes, and introducing constraints that increase the threshold of accuracy into the design (for a detailed discussion, see Sec. 4.2). Due to the relevance of this problem in biomedical fields, a conspicuous number of both mathematical and algorithmic methods have been developed for the design and selection of hybridization probes. Specifically, exact approaches and heuristic algorithms

have been broadly studied for the *Non-Unique Probe Selection Problem*; a detailed description of the approaches proposed in literature is provided in Section 4.3.

Then, in Sections 4.4 and 4.5, two new methods for the *Non-Unique Probe Selection Problem* are presented [POP11b]. The first is a canonical Monte Carlo algorithm that implements a heuristic reduction phase (see Sec. 4.4), aiming to minimize the number of probes required in the final design. The choice of applying a heuristic approach is mainly due to the complexity of the problem, that makes impracticable the construction of a feasible solution within a reasonable computational effort.
Starting from the results obtained by this method, a new combinatorial optimization approach, called Space Pruning Monotonic Search, is proposed [POP11b] (see Sec. 4.5). This new approach combines the quality guarantee provided by deterministic methods, with the exploring ability of heuristic procedures. In fact, while the exact method allows one to find solutions with a guarantee of optimality, the heuristic phase prunes the search space in order to obtain a feasible solution within a reasonable amount of time and computational resources.
Experiments have been conducted by using both artificial and real data sets, in order to assess the effectiveness of the approaches developed. The experimental results have been compared with the state-of-the-art methods; it turns out that both the Monte Carlo and the Space Pruning Monotonic Search represent promising approaches and, in particular, the Space Pruning Monotonic Search clearly outperforms the other methods proposed for the problem.

## 4.2 The Non-Unique Probe Selection Problem

The choice of the oligonucleotide probe sets plays an important role in hybridization experiments: the optimal probe set should contain a minimal number of oligonucleotides in order to minimize the number of hybridization experiments, since this affects the experimental costs [BCDV$^+$01]. Due to the importance of performing an efficient design and selection, this subject has attracted increasing interest in the last twenty years. Therefore, a plethora of approaches has been proposed in literature, ranging from specific software tools and web-based applications to select oligonucleotides having

specific characteristics, such as low similarity, absence of secondary structure, specific length [RHZ02, WS03, ELD03, BZJ+03, MSG02, TNK+03, NWK03, GS04, CHMS04, RCD+04, Nor05], to algorithms that optimize the design [KS02, LDB+96, SL03, LHZ05, RHMP05, CGF+05].
The Probe Selection Problem involves the selection of a small set of probes to analyze a population of unknown clones [Rag07]. The basic (binary) Probe Selection Problem can be formally stated as follows [BCDV+01]:

**Definition 4.2.1.** *Let $C = \{c_1, ..., c_m\}$ be a set of unknown clones (or targets), and $P = \{p_1, ..., p_n\}$ a set of oligonucleotide probes of preselected length $l$. The objective is to find a smallest set $P$ of probes such that any two distinct clones $c, d \in C$ are distinguished by at least one probe $p \in P$, where a probe $p$ is said to distinguish two clones $c, d$ if it is a substring of exactly one of $c$ or $d$.*

Finding probes that are unique through hybridization experiments is difficult, especially when targets have a high degree of similarity, as in the case of closely related viruses. An alternative approach consists in the selection of *non-unique* probes, characterized by the hybridization to more than one target. In addition to the intrinsic complexity of this task, the presence of errors can affect the results of the experiments: false hybridizations can occur (*false positive*), or experiments might not report their presence (*false negative*). A remedy is introducing redundancy into the design, by requiring that targets have to be separated by more than one probe (*separation constraint*), and that each target has to hybridize to more than one probe (*coverage constraint*). The formal definition of the *Non-Unique Probe Selection Problem* (*NUPS*, for short) requires some additional definitions.

**Definition 4.2.2** (Target-Probe Incidence Matrix)**.** *Given $m$ target sequences $t_1, ..., t_m$, and $n$ candidate probes $p_1, ..., p_n$, a target-probe incidence matrix $H = (h_{ij})$ is defined by $h_{ij} = 1$ if target $t_i$ hybridizes to probe candidate $p_j$, and $h_{ij} = 0$ otherwise.*

An example of target-probe incidence matrix can be found in Tab. 4.1.

**Definition 4.2.3** (Probe Coverage)**.** *Given $m$ target sequences $t_1, ..., t_m$, and $n$ candidate probes $p_1, ..., p_n$, a probe $p_j$, $1 \leq j \leq n$, is said to be covered by the target $t_i$, $1 \leq i \leq m$, if the target $t_i$ hybridizes to the probe $p_j$, or, equivalently, if $h_{ij} = 1$.*

| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ |
|---|---|---|---|---|---|---|
| $t_1$ | 1 | 1 | 0 | 1 | 0 | 1 |
| $t_2$ | 1 | 0 | 1 | 0 | 0 | 1 |
| $t_3$ | 0 | 1 | 1 | 1 | 1 | 1 |
| $t_4$ | 0 | 0 | 1 | 1 | 1 | 0 |

Table 4.1: Target-probe incidence matrix

**Definition 4.2.4** (Probe Separation)**.** *Given $m$ target sequences $t_1, ..., t_m$, and $n$ candidate probes $p_1, ..., p_n$, a probe $p_j$, $1 \leq j \leq n$, is said to distinguish or separate two target sequences $t_a$ and $t_b$, if it is a substring of exactly one of $t_a$ or $t_b$, i.e. if $|h_{aj} - h_{bj}| = 1$.*

As an example, given the targets $t_a = AGGCAATT$ and $t_b = CCATATT$ $GG$, for the probe $p_j = GCAA$, $h_{aj} = 1$, $h_{bj} = 0$; therefore $p_j$ distinguishes $t_a$ and $t_b$. On the contrary, for the probe $p_k = ATT$, we have $h_{ak} = h_{bk} = 1$, so that it follows $p_k$ does not distinguish the target pair $t_a$, $t_b$.
Informally, the goal of the *Non-Unique Probe Selection Problem* is to select a minimum set of probes such that the presence or absence of every single target can be univocally determined. In order to reduce the presence of experimental errors, each pair of targets should be distinguished by at least $h_{min}$ probes; these requirements are called *Hamming distance constraints*[1] or *separation constraints*. The *coverage constraint* requires that each target has at least $c_{min}$ probes hybridizing to it.
Let us consider the following example: in Table 4.1, if $h_{min} = c_{min} = 1$, and assuming that only a target among $t_1, t_2, t_3, t_4$ is present in the sample, a minimal solution is represented by the set of probes $\{p_1, p_2, p_3\}$. In fact, for target $t_1$, $p_1$ and $p_2$ hybridize, while $p_3$ does not; for target $t_2$, $p_1$ and $p_3$ hybridize, while $p_2$ does not; probes $p_2$ and $p_3$ hybridize to target $t_3$, while $p_1$ does not; finally, only probe $p_3$ hybridize to $t_4$.
Nevertheless, our assumption that only a single target is present in the sample is not realistic. The above solution containing the probes $p_1, p_2, p_3$, results in the hybridization of all probes if both the targets $t_1$ and $t_2$ are in our sample, and it is not possible to distinguish the case where the pair $(t_1, t_2)$ is present,

[1]We recall that the Hamming distance between two strings of equal length is defined as the number of positions at which the strings differ.

from the case where $t_3$ is also in the sample.
If we consider now the case where $h_{min} = c_{min} = 2$, any of the pairs $(t_1, t_3)$, $(t_1, t_4)$, $(t_2, t_3)$, $(t_3, t_4)$ hybridize to every probe in $\{p_2, p_3, p_5, p_6\}$; therefore, we can infer the presence of target $t_3$, but we cannot determine which other target is present. These considerations show that the *Non-Unique Probe Selection* is a difficult problem.
In particular, it can be proved that the problem is NP-hard using a reduction from the set cover problem, as a special case for $h_{min} = 1$ and $c_{min} = 0$ [KRS$^+$04, KRS$^+$07].

## 4.3 State-of-the-art Methods

Due to its importance in genomic applications, the *Non-Unique Probe Selection Problem* has been broadly studied, and a great number of mathematical and algorithmic approaches have been proposed. In particular, we distinguish between exact and heuristic methods. In the following section, we introduce the state-of-the-art methods for the problem, by describing both the exact and heuristic approaches presented in literature.

### 4.3.1 Exact methods

Many combinatorial formulations of the *Non-Unique Probe Selection Problem* have been presented in literature. In [KRS$^+$04, KRS$^+$07], after a preliminary reduction of the numbers of candidate probes, the problem is modeled as a variation of a set-cover integer linear programming.
Let $N = \{p_1, ..., p_n\}$ denote the set of candidate probes, $M = \{t_1, ..., t_m\}$ denote the set of targets, and $P = \{(i, k) | 1 \leq i < k \leq m\}$ denote the set of all combinations of target indices. Let $x_j$, $j \in N$, be the binary-valued decision variables, with $x_j = 1$ if the probe $j$ is chosen, 0 otherwise.
The problem is formulated as follows:

$$\min \sum_{j=1}^{n} x_j \qquad \text{(master ILP)} \tag{4.1}$$

subject to

$$\sum_{j=1}^{n} h_{ij}x_j \geq c_{min} \qquad \forall i \in M \tag{4.2}$$

$$\sum_{j=1}^{n} |h_{ij} - h_{kj}|x_j \geq h_{min} \qquad \forall (i,k) \in P \tag{4.3}$$

$$x_j \in \{0,1\} \qquad \forall j = 1, ..., n. \tag{4.4}$$

The objective function (4.1) minimizes the number of probes; the constraints in (4.2) require that at least $c_{min}$ selected probes hybridize to each target, and (4.3) represents the Hamming distance constraints. Finally, (4.4) restricts the variables to binary values.
When it is not possible to ensure $h$-separation for all target pairs with the given set of probes, the solutions of the above ILP formulation is empty. As a remedy, a large number $l = m \cdot h$ of unique virtual probes is added to the initial set of available probes. It is important to note that such "artificial" probes are chosen only if the separation constraint does not hold the original set of candidate probes. In this scenario, the objective function coefficients of the virtual probes are set to a large number $M$; the original master ILP is hence reformulated as follows:

$$\min \sum_{j=1}^{n} x_j + M \sum_{k=n+1}^{n+l} x_j, \tag{4.5}$$

with $n$ replaced by $n+l$ in the constraints of the original master ILP formulation.
The master ILP formulation presented in (4.1 )- (4.4) guarantees separation between pairs of single targets. In some circumstances, however, separation between groups of targets is desirable, since this approach can take into account cross-hybridization and error tolerance. A group-testing approach for this case has been already introduced in [STR03], where separation is defined also among pairs of small target groups.
For the mathematical formulation, group-separability is considered in a new model, the slave ILP. Given a set of targets $S$, let $\omega_j^S = \max_{i \in S} h_{ij}$ the signature of $S$, resulting from applying the logical $OR$ to the rows in $S$, with $j = 1, ..., n$. $S$ and $T$ are *d-separable* if and only if the Hamming distance between $\omega^S$ and $\omega^T$ is at least $d$. A cutting-plane approach is proposed to

solve this problem: the idea is to iteratively construct a most violated group constraint by looking at the current solution.
Let $x^*$ be a solution vector, and $X = \{j | x^*_j = 1\}$ the index set of probes included in $x^*$. For a target set $S$, let $\omega^S_{|X}$ be the restriction of $\omega^S$ to the columns in $X$. The following slave ILP is solved to find target groups $S$ and $T$ such that the Hamming distance between $\omega^S_{|X}$ and $\omega^T_{|X}$ is less or equal than $d$:

$$\max \sum_{j \in X} (\sigma^0_j + \sigma^1_j) \qquad \text{(slave ILP)} \tag{4.6}$$

subject to

$$\sigma^0_j \leq 1 - s_i \qquad \forall (i,j) \in M \times X \qquad \text{with } h_{ij} = 1 \tag{4.7}$$
$$\sigma^0_j \leq 1 - t_i \qquad \forall (i,j) \in M \times X \qquad \text{with } h_{ij} = 1 \tag{4.8}$$
$$\sigma^1_j \leq \sum_{i \in M} h_{ij} s_i \qquad \forall j \in X \tag{4.9}$$
$$\sigma^1_j \leq \sum_{i \in M} h_{ij} t_i \qquad \forall j \in X \tag{4.10}$$
$$\sum_{i \in M} s_i \geq 1 \tag{4.11}$$
$$\sum_{i \in M} t_i \geq 1 \tag{4.12}$$
$$s_i + t_i \leq 1 \qquad \forall i \in M \tag{4.13}$$
$$\sum_{i \in M} s_i \leq c \tag{4.14}$$
$$\sum_{i \in M} t_i \leq c \tag{4.15}$$
$$0 \leq \sigma^0_j \leq 1 \qquad \forall j \in X \tag{4.16}$$
$$0 \leq \sigma^1_j \leq 1 \qquad \forall j \in X \tag{4.17}$$
$$s_i \in \{0,1\} \qquad \forall i \in M \tag{4.18}$$
$$t_i \in \{0,1\} \qquad \forall i \in M \tag{4.19}$$

Variables $\sigma^0_j$ and $\sigma^1_j$ model the similarity between $S$ and $T$ for the probe $j$: $\sigma^0_j = 1$ if and only if $\omega^S_j$ and $\omega^T_j$ are equal to zero, $\sigma^1_j = 1$ if and only if both variables are equal to 1. Constraints (4.7) and (4.8) express that $\sigma^0_j = 0$ if $S$

or $T$ contain a target that hybridizes to probe $j$; due to the constraints (4.9) and (4.10), $\sigma_j^1 = 1$ if at least one target in both $S$ and $T$ hybridizes to probe $j$.
Finally, constraints (4.11) and (4.12) avoid that $S = \emptyset$ and $T = \emptyset$, while (4.13) and (4.14) control the size of $S$ and $T$.

A new formulation that does not require virtual probes is modeled in [KRS+07]. Here, the maximal possible separation between two groups of targets $S$ and $T$ is defined as:

$$h(S,T) = \sum_{j \in N} |\omega_j^S - \omega_j^T|. \tag{4.20}$$

Therefore the following model is introduced:

$$\min \sum_{j=1}^{n} x_j \tag{4.21}$$

subject to

$$x_j \in \{0,1\} \qquad \forall j \in N \tag{4.22}$$

$$\sum_{j \in N} |\omega_j^S - \omega_j^T| \cdot x_j \geq \min\{d, h(S,T)\} \qquad \forall S, T \subseteq M, \tag{4.23}$$

$$|S| \leq c, \qquad |T| \leq c, \qquad S \neq T.$$

Inequalities (4.23) ensure that coverage constraints are satisfied; since the number of constraints in (4.23) is exponential, a branch-and-bound approach is applied to solve the relaxed version of the problem.

In [DTMW08], an algorithm based on the previous ILP formulations is proposed to solve the *Non-Unique Probe Selection Problem* using a $d$-disjunct matrix.
A matrix is $d$-disjunct if and only if the union of any $d$ columns does not contain any other column, formally:

**Definition 4.3.1** ($d$-disjunct matrix)**.** *$H$ is a $d$-disjunct matrix if and only if for any $(d+1)$ columns $t_0, t_1, ..., t_d$, there exists a row $p_i$ such that $H[i,0] = 1$ and $H[i,j] = 0$, $\forall j = 1, ..., d$.*

Row $p_i$ is said to cover the pair $(t_0, \langle t_1, ..., t_d \rangle)$. The problem is hence reformulated as follows:

**Problem 4.3.2** (Minimum $d$-Disjunct Submatrix (MIN-$d$-SD))**.** *Given $m$ non-unique probe candidates and an $m \times n$ target-probe incidence matrix $M$, select a minimum set of the probe candidates such that the $h \times n$ submatrix $H$ is $d$-disjunct, where $h \leq n$.*

Of course, also the new defined problem is NP-hard [TZ09]. This problem is addressed by reformulating the ILP models proposed in [KRS$^+$07]. The first new ILP formulation is based on the following definition:

**Definition 4.3.3.** *$H$ is called $(d, k)$-disjunct if each column has at least $(k + 1)$ 1-entries not contained in the union of other $d$ columns.*

The following mathematical formulation is used to construct a $(1, d-1)$-disjunct submatrix.

$$\min \sum_{i=1}^{m} x_j \tag{4.24}$$

subject to

$$\sum_{i \in M} |M_{ij} - M_{ij}M_{ik}| x_i \geq d \qquad \forall j, k \in N, \qquad j \neq k \tag{4.25}$$

$$x_j \in \{0, 1\} \qquad \forall i \in M, \tag{4.26}$$

where $M$ is the target-probe incidence matrix; $m$ and $n$ are, respectively, the number of probes and targets; $x_i = 1$ if probe $p_i$ is chosen for the submatrix $H$, otherwise is 0. The first constraint guarantees that any column in $M$ has at least a number $d$ of 1-components not contained in any other column.
A second ILP formulation recalls the slave ILP proposed in [KRS$^+$07]; it finds the target sets $R$ and $S$ that violate $d$-disjunctness.
In order to detect errors in hybridization experiments, a new ILP problem is modeled, that is able to identify at most $d$ targets when at most $k$ errors occur, by constructing a $(d, 2k)$-disjunct matrix [DH06].

$$\min \sum_{i=1}^{m} x_j \tag{4.27}$$

subject to

$$\sum_{i \in M} |M_{ij} - M_{ij}M_{ik}|x_i \geq d + 2k \qquad \forall j,k \in N, \qquad j \neq k \tag{4.28}$$

$$x_j \in \{0,1\} \qquad \forall i \in M. \tag{4.29}$$

The algorithm for the error tolerance case uses this formulation to construct the $(1, d+2k-1)$-disjunct matrix, and the second one to find the target sets that violate $(d, 2k)$-disjunctness.

### 4.3.2 Heuristic methods

Though exact methods are able to locate solutions with a guarantee of optimality, these approaches require an exact knowledge of the domain; moreover, in some cases, finding a solution via integer programming software is intractable, due to the large number of variables and constraints within the problem [MLO+04]. Heuristic algorithms allow to overcome these difficulties, by implementing search strategies that speed up the location of optimal solutions, even if without any guarantee of optimality.

A two-phases heuristic algorithm for the *Non-Unique Probe Selection Problem* has been presented in [MPR07]. It consists of a construction phase, which builds a feasible solution when possible, and a reduction phase that reduces the number of probes while maintaining feasibility. To begin with, candidate probes are selected according to the number of targets to which each probe binds, until the minimum coverage constraint is satisfied for each target. The algorithm next focuses on satisfying the minimum Hamming distance constraint: probes that are not included in the current solution are sorted in decreasing order by the number of target pairs they separate; then probes are selected from the ordered list and added to the solution until the number of Hamming distance violated constraints is reduced to a predefined threshold. The remaining probes are found using a local search. The reduction phase iteratively deletes two probes at a time and verifies if the solution is still feasible, or if it can be made feasible by adding just one probe to the solution.

A cutting plane algorithm is presented in [RSP07]: this method relaxes a large subset of Hamming distance constraints, in order to find and improve the lower bound on the number of probes required in an optimal solution. Relaxed constraints are reinstated only if it is needed to maintain feasibility. Results provided by this approach are compared to the values computed by the heuristic algorithm implemented in [MPR07] and the ILP approach presented in [KRS+04]: the cutting-plane algorithm outperforms the two approaches, in particular it is able to find solutions that are about 20% better than the previous best ones provided by the other two methods.

A model-based approach is discussed in [WN07], where a greedy heuristic uses a Bayesian model [Pel05] for guiding the search towards the probes that satisfy separation and coverage constraints. In general, a Bayesian optimization algorithm generates an initial population of solutions randomly, according to a uniform distribution. Therefore, the current population is updated for a number of iterations, by selecting the best solutions using methods inspired to Genetic Algorithms. This allows one to construct a Bayesian network representing the population of promising solutions, and new solutions are generated by sampling the defined probabilistic model. The new candidate solutions replace some of the old ones in the original population, in order to obtain in-silicon evolutionary pressure.
The model-based approach proposed in this work is similar to a Bayesian algorithm, but it does not consider any evolution mechanism. It represents a modification of the heuristic presented in [MPR07]: the idea is to prefer probes that appear in the largest number of minimal sets, by distinguishing between *good* and *bad* ones. Good probes are those with the highest degree of contribution to minimal solutions, whereas bad probes have the lowest degree of contribution. In order to include good probes in a feasible candidate solution, a coverage and a separation probabilistic models are implemented as follows.
Given the target-probe incidence matrix $H$, a candidate probe set $P = \{p_1, ..., p_n\}$ and a target set $T = \{t_1, ..., t_m\}$, let the function $cov_{drc} : P \times T \rightarrow [0, 1]$ be defined as follows:

$$cov_{drc}(p_j, t_i) = h_{ij} \times \frac{c_{min}}{|P_{t_i}|}, \qquad p_j \in P_{t_i}, \qquad t_i \in T \tag{4.30}$$

where $P_{t_i}$ is the set of probes hybridizing to target $t_i$. Notice that $cov_{drc}(p_j, t_i)$ is the amount that $p_j$ contributes to satisfy the coverage constraint for target

$t_i$. Additionally we have $0 \leq cov_{drc}(p_j, t_i) \leq 1$. The coverage function $C_{drc} : P \rightarrow [0, 1]$ is then defined as follows:

$$C_{drc}(p_j) = \max_{t_i \in T_{p_j}} \{cov_{drc}(p_j, t_i) \quad | \quad 1 \leq j \leq n\}, \tag{4.31}$$

where $T_{p_j}$ is is the set of targets covered by $p_j$. $C_{drc}(p_j)$ is the maximum amount that $p_j$ can contribute to satisfy the minimum coverage constraints. Function $C_{drc}$ favors the selection of probes that $c_{min}$-cover targets $t_i$ that have the smallest subsets $P_{t_i}$; these are the essential or near-essential covering probes. In particular, $C_{drc}$ guarantees the selection of near-essential covering probes that $c_{min}$-cover *dominated targets*, where $t_i$ dominates $t_k$ if $P_{t_k} \subset P_{t_i}$. Similarly, the function $sep_{drc} : P \times T^2 \rightarrow [0, 1]$ is defined as

$$sep_{drc}(p_j, t_{ik}) = |h_{ij} - h_{kj}| \times \frac{h_{min}}{|P_{t_{ik}}|}, \qquad p_j \in P_{t_{ik}}, \qquad t_{ik} \in T^2 \tag{4.32}$$

where $P_{t_{ik}}$ is the set of of probes separating target-pair $t_{ik}$; $sep_{drc}(p_j, t_{ik})$ is what $p_j$ can contribute to satisfy the separation constraint for target-pair $t_{ik}$. We have $0 \leq sep_{drc}(p_j, t_{ik}) \leq 1$. The separation function $S_{drc} : P \rightarrow [0, 1]$ is:

$$S_{drc}(p_j) = \max_{t_{ik} \in T^2_{p_j}} \{sep_{drc}(p_j, t_{ik}) \quad | \quad 1 \leq j \leq n\}, \tag{4.33}$$

where $T^2_{p_j}$ is the set of target-pairs separated by $p_j$. $S_{drc}(p_j)$ is the maximum amount that $p_j$ can contribute to satisfy the minimum separation constraints. Function $S_{drc}$ favors the selection of essential or near-essential probes that $h_{min}$-separate *dominated target pairs*; these probes $h_{min}$-separate targets-pairs $t_{ik}$ that have the smallest subsets $P_{t_{ik}}$.
Now, cover and selection functions are combined together into the selection function, that allows to choose the minimum number of probes such that coverage and separation constraints are satisfied:

$$D_{drc}(p_j) = \max\{(C_{drc}(p_j), S_{drc}(p_j)) \quad | \quad 1 \leq j \leq n\}. \tag{4.34}$$

$D_{drc}(p_j)$ represents the degree of contribution of $p_j$, that is, the maximum amount required for $p_j$ to satisfy all the constraints; $D_{drc}$ ensures that all essential probes $p_j$ will be selected for inclusion in the subsequent candidate solution, since $C_{drc}(p_j) = 1$ or $S_{drc}(p_j) = 1$.
The model-based algorithm uses the above introduced functions to create

a feasible set of probes, according to the constraints of coverage and separation. This algorithm consists of three phases: the initialization phase creates an initial solution; since this solution can be infeasible, the construction phase repeatedly inserts high-degree probes into the initial solution in order to maintain feasibility; finally, the reduction phase removes low-degree probes to reduce the cardinality of the solution set. This heuristic is called *Dominated Row Covering* heuristic. Results provided by the described model are compared with the ones obtained by applying the ILP method [KRS+04] and the heuristic algorithm presented in [MPR07]: the Bayesian algorithm obtains comparable results for the artificial data, and better results on real instances.

The selection strategy introduced in [WN07] has been combined with an evolutionary approach in [WNGR08], where a Genetic Algorithm, already proposed for the Set Cover Problem in [BC96], is adapted to the *Non-Unique Probe Selection Problem*. After the creation of a random population, two solutions are selected, by using fitness scaling and binary tournament. Therefore, a fusion crossover is used to create a new solution, to which a mutation operator is applied. In order to guarantee the feasibility, the heuristic presented in [WN07] is applied to the generated solution. At this point, a new solution has been generated from the pool of "parents", and it is used to replace a randomly selected solution with an above-average fitness in the population. This process will be iterated until a stopping criterion is met.
Each solution is modeled as an $n$-bit binary string $s = s_1...s_n$, where $s_j = 1$ if $p_j \in S$ and $s_j = 0$ if $p_j \notin S$, $1 \leq j \leq n$. The fitness of an individual $s$ is directly related to its objective value, which corresponds to the number of probes in its associated subset $S$. That is, the fitness of $s$ is defined as

$$f(s) = \sum_{j=1}^{n} s_j. \tag{4.35}$$

To create a new population from the current one, crossover and mutation operators are applied. The crossover operator used to create a new individual from two parental ones is a generalized fitness-based crossover: it considers the differences between the parents and is more capable of generating new solutions when the parents are similar; also, the fittest parent influences more the "child" solution. Let $f_{P_1}$ and $f_{P_2}$ be the fitness values of the parents $P_1$ and $P_2$ respectively, and let $C$ denote the child solution. Then for $1 \leq j \leq$

$n$, $C_j = P_{1j} = P_{2j}$, provided that $P_{1j} = P_{2j}$; otherwise $C_j = P_{1j}$, with probability $p = f_{P_2}/(f_{P_1} + f_{P_2})$, and $C_j = P_{2j}$, with probability $1 - p$.
After the crossover, the algorithm applies the mutation operator by randomly altering a number of bit positions, in order to introduce diversity and prevent the stagnation into local optima. The number of positions to mutate for a given solution depends on the mutation rate, which is correlated to the convergence rate of the algorithm; it is defined as:

$$B = \left\lceil \frac{m_f}{1 + \exp \frac{-4m_g(t-m_c)}{m_f}} \right\rceil, \tag{4.36}$$

where $t$ is the current number of child solutions generated, $m_f$ specifies the final stable mutation rate, $m_c$ is the number of solutions that should be generated such that the mutation rate is $\frac{m_f}{2}$, and $m_g$ specifies the gradient at $t = m_c$. The value of $m_f$ is user-defined and the values of $m_c$ and $m_g$ are problem-dependent parameters.
The initial population is generated with a high probability of being feasible, by putting

$$s_{cj} = \begin{cases} 1 & \text{if } r \leq D_{drc}(p_j) \\ 0 & \text{otherwise} \end{cases} \quad 1 \leq j \leq n \tag{4.37}$$

where $r \in [0, 1]$ is randomly chosen.
If the solution is infeasible, a heuristic feasibility operator, consisting of the Construction and Reduction Phase of the model presented in [WN07], is applied to maintain feasibility. In order to replace solutions in the current population, a steady-state replacement strategy is implemented: a new solution replaces a randomly chosen individual of the population having a higher fitness value. This strategy is elitist since the best solutions are picked and they replace always the worst ones; in this way the population will converge quicker. This method allows one to create high quality solutions; in fact it outperforms all the previous implemented methods, by constructing a probe set with minimum cardinality.

Though the Dominated Row Covering heuristic presented in [WN07] guarantees the selection of near-essential probes that $c_{min}$-cover and $h_{min}$-separate dominated targets, it is not able to distinguish among targets having the same values for $C_{drc}$ or $S_{drc}$. To address this task, a Dominated Probe Selection

heuristic has been introduced in [WNR08]. Here, the coverage and separation functions penalize each probe $p$ by an amount that is proportional, respectively, to the number of targets that it covers or separates.

In order to guarantee separation not only between target pairs, but also between pairs of small groups of targets, a subset selection criterion has been introduced in [WNR08]. This criterion is used by a sequential forward algorithm [PNK94] to select the best subset of probes; in particular, it iteratively selects the probe that maximizes the Dominant Probe Selection criterion, and adds it to the current probe set. When a feasible solution is found, a reduction phase based on [WN07] is applied. This method performs better than the approaches proposed in [MPR07, STR03]; nevertheless, the Genetic Algorithm proposed in [WNGR08] and the cutting plane method developed in [RSP07] provide better results for all the tested datasets.

In [NRWG10], the two heuristics presented so far, the dominated row covering and the dominated probe selection, have been modified in order to consider, at each step, the knowledge about the probes that are already selected in the previous iterations. In fact, these methods compute coverage and separation values during the initialization phase, and their values remain unchanged for the rest of the computation. Three new heuristics are introduced: the Dynamic Dominated Row Covering Heuristic, the Dynamic Dominant Probe Selection Heuristic and the Normalized Dynamic Dominant Probe Selection Heuristic. The first, takes into account the information about selected probes: if a probe $q$ is selected for a given target or target-pair, coverage and separation are re-computed only for those rows affected by this selection.

In a further work [GGWN09], the dominated row covering heuristic, the dominated probe selection heuristic, and a new one, the Sum of Dominated Row Covering, have been combined with a Bayesian optimization algorithm [LL02]. The new heuristic introduced in [LL02] is capable to distinguish among probes that have the same score for the coverage of the dominated targets, and the same score for the separation of the dominated target pairs. This new approach outperforms the other methods in some test instances.

In [GGWN10], a multiobjective optimization method has been combined with the Bayesian algorithm presented in [GGWN09] to solve the *Non-Unique Probe Selection Problem* for the multiple targets case. The two objectives for this problem are minimizing the number of probes involved in a solution and maximizing the ability of recognizing multiple targets. In order to evaluate the fitness functions, a modified version of the Weighted Average Ranking (WAR, for short) is computed. In WAR, each objective is evaluated separately and the fitness values of the solutions for each objective are sorted. Solutions are then ranked according to their fitness, finally the ranks obtained by sorting each list of objectives are averaged. This approach guarantees successful results for the simple cases of five and ten targets in the sample, and good results for the case of fifteen in almost all the instances considered in the dataset. When twenty targets are considered, it is able to find good solutions for some of the real and artificial datasets.

## 4.4 Monte Carlo algorithm with Heuristic Reduction

The first new method presented in this chapter for the *Non-Unique Probe Selection Problem* is a canonical Monte Carlo algorithm with Heuristic Reduction (MCHR, for short) [POP11b]. Monte Carlo methods [MRR+53, MU49] are stochastic approaches used to model several phenomena in various fields, from physics to engineering, from biology to finance.

The choice of developing a Monte Carlo approach for the *NUPS* problem has been guided by the ability of Monte Carlo methods to perform an extensive search in the solution space, by applying a probabilistic mechanism that avoids local minima. Specifically, this is implemented by the possibility of accepting not only transitions that improve the value of the objective function, but also moves that can deteriorate the solution quality.

The first step in designing our Monte Carlo approach consists in modeling the *NUPS* as an optimization problem, by defining the cost function. Since the objective of the *Non-Unique Probe Selection Problem* is to minimize the number of probes included in the final design, an optimal solution should contain the minimum number of probes such that the presence of each target can be univocally determined, while the separation and coverage constraints are verified. In particular, the proposed Monte Carlo algorithm builds a feasible

initial solution, by using the Dominated Row Covering Heuristic proposed in [WNGR08]: the idea is to prefer the choice of probes that appear in the largest number of minimal sets, by distinguishing between "good" and "bad probes". "Good probes" are those with the highest degree of contribution to minimal solutions, whereas "bad probes" are those with the lowest degree of contribution to minimal solutions. In other words, this heuristic favors the selection of probes that cover and separate many target sequences, where these targets hybridize to the minimum number of probes such that the coverage constraint holds (i.e., the number of probes that cover each target is as close as possible to the value $c_{min}$), and the number of probes separating those targets is the minimum allowed (i.e., the number of probes that separate each target pair is as close as possible to the value $h_{min}$).
The choice of the Dominated Row Covering Heuristic is mainly due to efficiency considerations [NRWG10]: though the other heuristic strategies presented in Sec. 4.3 [WN07, WNGR08, WNR08, GGWN09, NRWG10] are able to locate promising solution, their computational complexity varies between $\mathcal{O}(n^2m^4)$ and $\mathcal{O}(n^8m^8)$, where $n$ is the cardinality of the probe set, and $m$ is the number of targets. On the other hand, the Dominated Row Covering Heuristic runs in $\mathcal{O}(n^2m^2)$ time. Therefore, the limited gain in terms of quality of the solutions seems not justify the computational cost needed to reach it.

In order to include the good probes in a feasible candidate solution, the selection function $D : P \rightarrow [0, 1]$ is computed for each probe, according to the Dominated Row Covering Heuristic. Given a set of $n$ probes $P = \{p_1, p_2, ..., p_n\}$, $D(p_j)$, $1 \leq j \leq n$, represents the degree of contribution of the probe $p_j$, that is, the maximum amount required for $p_j$ to satisfy all the constraints. The higher is the value of $D(p_j)$, the better is the quality of probe $p_j$.

MCHR starts by computing the value of the heuristic $D$ for each probe; then an initial solution $s_{initial}$ is randomly constructed with a high probability of being feasible [WNGR08], since each probe is chosen probabilistically according to its $D$ value. In this way, the choice of "bad" probes is discarded in favour of a design characterized by a "good" initial solution.
If this solution is infeasible, a heuristic feasibility operator [WNGR08] is applied to maintain feasibility. This operator consists of two phases: first, a construction phase iteratively chooses, among the probes not included in

the current solution, those having the highest value for the Dominated Row Covering Heuristic. These probes are then added to the incumbent solution, until a feasible solution is built.
Therefore, a reduction phase tries to reduce the number of probes, by deleting those with the lowest value of the heuristic since they poorly contribute to good solutions, if this procedure does not affect the feasibility.
The pseudocode of the Monte Carlo algorithm with Heuristic Reduction is presented in Alg. 3. Once a feasible solution is constructed, at each iteration

**Algorithm 3** Pseudocode of the MCHR algorithm.

$s_{cur} \leftarrow s_{initial}$
$f_{cur} \leftarrow f_{initial}$
**while** $\neg$ stopping criterion **do**
  $s_{new} \leftarrow$ random_reduction$(s_{cur}, \rho)$
  $\Delta \leftarrow \min(1, \exp(-\beta * (f_{new} - f_{cur}))$
  **if** $(f_{new} < f_{cur})$ **then**
    $s_{new} \leftarrow s_{cur}$
  **else**
    **if** $((f_{new} \geq f_{cur}) \wedge (random < \Delta))$ **then**
      $s_{new} \leftarrow s_{cur}$
      $EI \leftarrow (\gamma * f_{cur})$
      **if** $((f_{cur} - EI) < f_{best})$ **then**
        $s_{new} \leftarrow$ greedy_reduction$(s_{cur}, \gamma)$
      **end if**
    **end if**
  **end if**
**end while**

of the main loop, a new solution is constructed from the current solution, by applying the *random reduction* procedure (line 4): this step selects a random number of probes to delete from the current solution. This number depends on a parameter $\rho$, representing the percentage of probes that we aim to discard from the current solution. Since this new solution might be infeasible, the heuristic feasibility operator is applied.
By deleting probes from the initial solution, a new solution is generated. If this new solution is better than the old one (line 6), i.e. contains less probes, it will become the new incumbent used by the next iteration of the algorithm (line 7).

We want to recall that Monte Carlo methods accept not only solutions that improve the value of the objective function, but also solutions that do not, in order to avoid the stagnation of the algorithm into a local optimum. In order to achieve this non-monotonic search behavior, a solution that is worse than its predecessor is accepted according to the following probability:

$$p = e^{(-\beta(f_{new} - f_{cur}))}, \tag{4.38}$$

where $f_{new}$ and $f_{cur}$ represent, respectively, the objective value of the new generated solution and of the "old" one, and $\beta \in (0, 1]$ is a parameter of the algorithm (line 5). The acceptance probability depends on the quality of the solution, so that good solutions have a higher probability to be accepted. When this happens, a new step of the algorithm is performed: the expected improvement (EI) is computed in line 11; this represents the number of probes to delete from the current solution, according to the input parameter $\gamma$.
Therefore, if it is possible to obtain a solution that contains less probes than the best one by deleting a fraction $\gamma$ of them (line 12), the *greedy reduction* procedure is performed (line 13). This step iteratively tries to delete the probes one by one; if this results in an infeasible solution, the probe is reinstated in the current solution.

## 4.5 Space Pruning Monotonic Search

Combinatorial optimization problems are usually characterized by very rugged and, frequently, ill-conditioned search space; in general, if the optimization procedure starts from a region of the search space which does not contain the optimal solution, it is difficult and computationally expensive to escape from this basin towards a new promising one. This process becomes extremely hard to accomplish especially when the dimension of the problem is relevant.

Deterministic and heuristic methods try to overcome these difficulties by introducing some globalization techniques; the paradigm of globalization strategies is to detect a search direction that achieves a descent in the value of a given objective function, in order to improve the likelihood of convergence when good initial solutions are not available [PSSW06].
Globalization methods can be classified into three main categories: stochastic, population based and multi-start strategies [KL08]. Stochastic approaches, like perturbed gradient [PSdC94] and simulated annealing [KGV83], incorporate probabilistic elements to iteratively compute a solution, and adjust

the parameters to improve its quality. Population-based strategies belong to the class of the nature-based optimization algorithms; they compute a population of solutions, which are combined to generate new candidate points. Examples of population-based methods are Genetic Algorithms [Hol92] and Ant-Colony Optimization [Dor92]. Multi-start methods are characterized by the use of a minimization algorithm, like gradient descent, generally coupled with a heuristic procedure that aims to improve the solution by sampling the search space [PSdC94, Mar03]; the output of the algorithm is the best overall solution found among the ones generated at each iteration.

By considering the field of deterministic methods, a common feature is the ability of efficiently locating an optimal solution, when starting from a promising region of the solution space; it implies the use of a promising initial iterate, for instance a user-provided solution, or pruning the solution space of the problem. A pruning technique is a procedure that provides a sub-region of the search space of the problem $P'$ starting from the original problem $P$, in order to reduce the computational burden needed to find an optimal solution, and to avoid the exploration of known basins. The name is inspired by gardening, in which pruning means to clip off branches on a tree, that corresponds to what we "ideally" perform on the search space.
Pruning techniques are commonly implemented in search algorithms, since they are able to enormously reduce the size of the original problem, and focus the search into promising basins of attraction. For instance, the bound step of any branch-and-bound algorithm realizes a pruning strategy, by reducing the search tree to a computationally manageable size.

Starting from this considerations, we developed a new method for the *Non-Unique Probe Selection Problem* [POP11b], which implements a pruning strategy to narrow the search space. The proposed new approach, called Space Pruning Monotonic Search (SPMS), is an iterative method which applies a pruning of the input problem $P$, and then uses a deterministic method to solve the pruned problem $P'$ (see Alg. 4). In particular, the pruning strategy allows to locate a promising solutions basin, while the power of the deterministic approach is exploited to efficiently solve the subproblem. We want to highlight that this method is extremely general, which means that any pruning and optimization method can be implemented to perform the search into the narrowed space created by the pruning procedure.

**Algorithm 4** Pseudocode of the SPMS algorithm.

```
1: P* ←pruning(P)                          ▷ Initial pruning
2: f* ←pruning(P*)                         ▷ Initial evaluation
3: while ¬ converged do
4:     P' ←pruning(P*)
5:     f' ←solve(P')
6:     if (f' < f*) then
7:         f* ← f'
8:         P* ← P'
9:     end if
10: end while
```

The algorithm is monotonic, as stated in its name; such monotonic property guarantees that only solutions representing an improvement of the objective value are accepted. In fact, many recent results in monotonic search methods have shown that this approach is extremely effective, especially when combined with heuristic solution perturbations and deterministic solvers [ACLS09, GLS09]. Specifically, the monotonic behavior prevents the choice of degenerate solutions, while the heuristic perturbation introduces changes into the current solution aiming to improve its quality.

Since the Space Pruning Monotonic Search introduced in this section deliberately represents a general framework to solve optimization problems, this method can be extended to take into account different strategies; for instance it can implement several accepting criteria, according to the specific problem addressed. Analogously, different convergence criteria can be defined, e.g. the attainment of a prefixed number of iterations or a quality threshold.
Additionally, many improvement techniques can be introduced in SPMS; as an example, in order to improve its effectiveness, our approach can be extended with a randomized pruning technique: by exploring different and, possibly, random subregions of the search space, this strategy might enhance the coverage of the whole landscape and hence lead to better solutions.

### 4.5.1 A SPMS approach for the *Non-Unique Probe Selection Problem*

We apply the SPMS approach introduced in the previous section for solving the *Non-Unique Probe Selection Problem*; in particular, an Integer Linear Programming (ILP) formulation is used to model the problem, and an ad-hoc pruning technique is proposed to narrow the search space. The ILP formulation adopted to define the problem has been first proposed in [KRS+04] (see Section 4.3.1); it is defined as follows:

$$\min \sum_{j=1}^{n} x_j \tag{4.39}$$

subject to

$$\sum_{j=1}^{n} h_{ij} x_j \geq c_{min} \qquad \forall i \in M \tag{4.40}$$

$$\sum_{j=1}^{n} |h_{ij} - h_{kj}| x_j \geq h_{min} \quad \forall (i,k) \in P \tag{4.41}$$

$$x_j \in \{0,1\} \quad \forall j = 1, ..., n \tag{4.42}$$

The objective function (4.39) aims to minimize the number of probes in the final design; the coverage constraints (4.40) require that at least $c_{min}$ selected probes hybridize to each possible target, and the separation constraints (4.41) ensure that each target-pair is separated by at least $h_{min}$ probes. The decision variables $x_j$ are binary (4.42): $x_j$, $1 \leq j \leq n$, is equal to 1 if the probe $p_j$ is chosen for the solution, otherwise is 0. When it is not possible to $h_{min}$-separate all pairs of targets with the initial set of probes, the solution of the above ILP is empty. In this case, a large number $l = m \cdot h_{min}$ of unique virtual probes is added. Such probes are chosen only if the separation constraint does not hold with the original set of candidate probes, by setting the objective function coefficients of the virtual probes to a large number $M$. The original objective function (4.39) is reformulated as:

$$min \sum_{j=1}^{n} x_j + M \sum_{k=n+1}^{n+l} x_j \tag{4.43}$$

where $n$ is replaced by $n+l$ in the constraints of the original ILP formulation.

Since the problem involves only binary variables, as a pruning technique we force the presence of good probes in our solution, by fixing the variables corresponding to these probes. In this way, starting from the original problem $P$, where the solution space contains the original set of probes, we obtain a new problem $P'$, that is a subproblem of the original problem $P$. $P'$ presents the same objective value and the same constraints of $P$, but the new search space is limited to the fixed variables.

Specifically, the Dominated Row Covering heuristic [WNGR08] is chosen as the strategy for selecting the probes to include into the pruned problem $P'$: once the value of this heuristic is computed for each probe, the decision variables corresponding to the probes having the highest value are fixed. This means that, if a probe represents an essential component, it is forced to be part of the solution by setting the value of its decision variable to 1.
In particular, a minimum threshold for the heuristic is determined, and among the probes that present a value equal or higher than this prefixed bound, some are selected by a random procedure and added to the current solution. The randomization strategy introduced is able to better explore the search space, by allowing the selection of probes that present a high degree of contribution to optimal set, but not necessarily the highest value for the heuristic. In this way, even if the algorithm maintains his monotonic behavior by choosing only solutions that improve the objective function value, the stagnation of the algorithm into local optima is avoided: the mechanism of choosing the probes *among* the ones having the highest value for the heuristic, in fact, prevents that the search gets trapped into a suboptimal solution. Moreover, this simple randomized strategy bypasses the complex computation implemented in [WNR08] for distinguishing among probes having the same value for the Dominated Row Covering heuristic, by avoiding a dramatic increment of the computational complexity of the algorithm.

Once the essential components are selected by the randomized procedure, a deterministic method is used to solve the pruned problem; the method verifies if the located solution is feasible, otherwise it adds probes until a feasible solution is constructed. The idea behind this approach is to reduce the computational effort of the deterministic solver by exploiting the power of the heuristic; additionally, the high quality of the partial solution identified in the previous step allows the deterministic method to add a minimum number of probes to obtain a feasible solution.

## 4.6 Dataset and Experimental Results

### 4.6.1 Datasets and Experimental Protocol

The experiments for the methods presented in Sections 4.4 and 4.5 have been conducted on the same group of data: it includes ten artificial ($a1, ..., a5$, $b1, ..., b5$), and three real datasets (*Meiobenthos, HIV-1* and *HIV-2*).
The artificial data sets and the Meiobenthos sequences have been first proposed in [KRS+04], while the instances *HIV-1* and *HIV-2* have been used in [Rag07].

To generate the artificial data, the Random Evolutionary FORest Model (REFORM) software was used with two different forest models. From each model, five independent test sets were generated. The test sets generated from the first model (*a1-a5*) contain 256 targets each; from the second one, five test sets (*b1-b5*), with 400 targets each, are constructed. The choice of these topologies has been made because the sets of target sequences constructed present a high degree of similarity, so that they cannot be easily separated by using unique probes.
The probe candidates for each of the ten artificial test sets were generated using the Promide software [Rah03].

For the real datasets, three groups of sequences are constructed. In particular, one group contains sequences from Meiobentos organisms, and the other two are HIV sequences.
Benthos are organisms that live in the sea floor. They are categorized according to their size; specifically, Meiobenthos have size between $100\mu m$ and $500\mu m$ [BBH99]. In order to obtain the target sequences, 1,230 28S rDNA sequences from different organisms present in the Meiobenthos were clustered in 149 groups. Clusters were representative of approximately 56% of all $Meiobenthos$ sequences [STR03]. A test set of 679 targets has been obtained by arbitrarily selecting a representative from each cluster. The corresponding probe set contained 15,139 probes.
The second and the third group of real data consisted of 200 each *HIV-1* and *HIV-2* sequences downloaded from the National Center for Biotechnology Information; as for the artificial instances, these sequences present high similarity [MPR07]. Candidate probes have been generated using Primer3 [RS00] with default input parameters. The default parameters include: length be-

tween 18 and 27 nucleotides, melting temperature between 57°C and 63°C, and GC content between 20% and 80% for each HIV sequence.

The properties of the datasets are presented in Table 4.2: the second and third columns are the number of targets and number of probes of each data set, respectively. Since most of the data sets used are not able to satisfy the minimum Hamming distance constraints for every pair of targets, even if the entire set of candidate probes is chosen for the solution, artificial probes are added to the final solution set. Column $|V|$ represents the number of such virtual probes.

In order to prevent the presence of false negative and false positive, redundancy is added into the design by requiring that each probe hybridize to more than one target, and that each target-pair is separated by more than one probe; this is addressed by setting the parameters $c_{min} = 10$ and $h_{min} = 5$.
The stopping criterion for the MCHR algorithm is established as the number of iterations, and it is set to 4000. The parameters $\rho$, $\beta$, and $\gamma$ are, respectively, 0.1, 0.4 and 0.03 and have been experimentally fixed.
The SPMS performs 10 iterations, where the pruned ILP problem is solved by using CPLEX version 12.1[2].

### 4.6.2 Experimental Results

The two new methods presented in the previous sections have been compared with the state-of-the-art approaches for the *Non-Unique Probe Selection Problem* [POP11b, POP11a]; Table 4.3 provides the experimental results, and it reports the number of probes included in the final solution ($|P_{min}|$) for the methods proposed in the literature and our approaches. In particular, in bold face we report the best solution for each instance of the problem.
By inspecting these results, the MCHR algorithm seems to not guarantee the best performances; however, it is possible to note that its solutions are close to the putative global optima for the $HIV-1$ and $HIV-2$ instances. An analysis of the second dataset shows that MCHR finds the same pu-

[2]http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

| Set | $\|T\|$ | $\|P\|$ | $\|V\|$ |
|---|---|---|---|
| a1 | 256 | 2786 | 6 |
| a2 | 256 | 2821 | 2 |
| a3 | 256 | 2871 | 16 |
| a4 | 256 | 2954 | 2 |
| a5 | 256 | 2968 | 4 |
| b1 | 400 | 6292 | 0 |
| b2 | 400 | 6283 | 1 |
| b3 | 400 | 6311 | 5 |
| b4 | 400 | 6223 | 0 |
| b5 | 400 | 6285 | 3 |
| M | 679 | 15139 | 75 |
| HIV-1 | 200 | 4806 | 20 |
| HIV-2 | 200 | 4686 | 35 |

Table 4.2: Properties of the datasets used for experiments.

tative optima obtained of the Bayesian Algorithm; this result is extremely good since BOA is largely regarded as the best algorithm so far.

Turning to the SPMS method, it is important to note that it is able to find the putative optimal solution for many datasets; in particular, it finds two new putative optima for the set $a4$ and $M$. While for the $a4$ dataset the new solution contains 3 less probes that the best known solution, for the $M$ dataset there is a reduction of 59 probes, which is extremely remarkable. Moreover, this approach is very fast to compute the solution, whereas many of the previous proposed methods, such as the Genetic, Bayesian and Monte Carlo algorithms are slow due to their intrinsic characteristics [WNGR08].
SPMS cannot find the best solution for the datasets $b_2$ and $b_5$; these sets are characterized by the presence of many probes with low or the same heuristic values; it seems that the Dominated Row Covering heuristic is not able to distinguish between them producing low quality solutions [WNGR08].
The SPMS algorithm is not able to find good solutions for the instances *HIV-1* and *HIV-2*. Since there are no results in literature found by ILP formulations for these datasets, we solved the ILP model [KRS+04] by using

CPLEX without any heuristic reduction step, in order to understand if this performance can be apportioned to the characteristics of our approach or to the ILP model proposed. From this analysis, we found out that the number of probes included in the final solution is equal to 1075 and 1027 for *HIV-1* and *HIV-2*, respectively, which are very different from the SPMS results. This suggests that the intrinsic characteristics of the datasets make the problem hard to solve with the proposed ILP formulation, even if this problem is mitigated by our method.

Finally, we performed a ranking of the state-of-the-art methods for the *Non-Unique Probe Selection Problem*; specifically, we assigned a score based on the number of putative optimal solutions found by each algorithm. By inspecting the results in Table 4.4, it is possible to note that SPMS represents the best approach for this biological combinatorial optimization problem; instead, the MCHR algorithm is not well ranked even if it is superior to four other approaches. Moreover, it seems that deterministic and hybrid-deterministic methods are more suitable for this problem, since they reach the first two positions in our ranking.

## 4.7 Conclusions and Future Directions

The complexity in designing and selecting optimal probes for hybridization experiments makes this subject an interesting problem from a computational perspective. In particular, a plethora of methods have been proposed in literature to address several issues related to the selection and design of probes, ranging from exact approaches to heuristic algorithms. After introducing the biological aspects of hybridization procedures, this chapter illustrates the state-of-the-art methods for the described problems.

In particular, the *Non-Unique Probe Selection Problem* represents a challenging problem from a biological and computational point of view. Two new heuristics approaches are introduced to tackle it: a canonical Monte Carlo algorithm and a new optimization method called Space Pruning Monotonic Search [POP11b]. This last method brings a new concept of problem simplification, which provides promising regions of the search space to deterministic optimization methods.
We performed an extensive set of computational experiments to assess the quality of the solutions and the robustness of these new methods on a well

| | GrdS [STR03] | ILP [KRS+04, KRS+07] | GDRM [MPR07] | DRC [WN07] | OCP [RSP07] | GA [WNGR08] | SFPS [WNR08] | BOA [GGWN09] | MCHR | SPMS |
|---|---|---|---|---|---|---|---|---|---|---|
| a1 | 1163 | 503 | 568 | 549 | 509 | **502** | 530 | **502** | 541 | **502** |
| a2 | 1137 | 519 | 560 | 552 | 494 | **490** | 516 | **490** | 497 | 494 |
| a3 | 1175 | **516** | 613 | 590 | 543 | 534 | 557 | 533 | 570 | 520 |
| a4 | 1169 | 540 | 597 | 579 | 539 | 537 | 557 | 537 | 550 | **536** |
| a5 | 1175 | **504** | 605 | 583 | 529 | 528 | 558 | 528 | 574 | 523 |
| b1 | 1908 | 879 | 961 | 974 | **830** | 839 | 883 | 834 | 907 | **830** |
| b2 | 1885 | 938 | 976 | 1013 | **842** | 852 | 890 | 846 | 963 | 862 |
| b3 | 1895 | 891 | 951 | 953 | **827** | 835 | 896 | 829 | 960 | **827** |
| b4 | 1888 | 915 | 1001 | 1019 | **873** | 879 | 920 | 875 | 976 | **873** |
| b5 | 1876 | 946 | 1022 | 1019 | **874** | 890 | 933 | 879 | 981 | 897 |
| M | 3851 | 3158 | 2336 | 2084 | 1962 | 1962 | 2036 | - | 2029 | **1903** |
| HIV-1 | - | - | 531 | 487 | 451 | **450** | 468 | **450** | 467 | 954 |
| HIV-2 | - | - | 578 | 506 | 479 | 476 | 492 | **474** | **474** | 949 |

Table 4.3: $|P_{min}|$ value of the proposed method.

| Rank | Method | Score |
|---|---|---|
| **1** | **SPMS** | **6** |
| 2 | OCP[RSP07] | 5 |
| 3 | BOA[GGWN09] | 4 |
| 4 | GA[WNGR08] | 3 |
| 5 | ILP [KRS$^+$04, KRS$^+$07] | 2 |
| 6 | MCHR | 1 |
| 7 | GrdS [STR03] | 0 |
| 7 | GDRM[MPR07] | 0 |
| 7 | DRC [WN07] | 0 |
| 7 | SFPS[WNR08] | 0 |

Table 4.4: Ranking of the state-of-the-art methods.

known set of benchmarks and real data. The analysis of the results shows that SPMS is the best approach for the problem, since it reaches the greatest number of optimal solutions.

We think that there is still a lot of work to be done on this emerging and interesting research field: firstly, target-group separation for the *Non-Unique Probe Selection Problem* represents a new field, that has not been well investigated yet.
From a biological point of view, it is important to note that the experiments conducted for the *Non-Unique Probe Selection Problem* are focused on identifying five targets in the sample; it can be interesting to extend the existing approaches for the case of more targets since this could be more realistic from a biological point of view. A new approach for the problem could concern the reformulation of separation and coverage constraints as a unique metric, since the resulting problem could be easier to solve. Moreover, the intrinsic characteristics of the datasets used for the experiments have not been well explored. Since they affect the computational results, it would be desirable taking into account the biological information contained in the sequences, in order to better explore the search space and deeply understand the results. Relative to the proposed algorithms, we believe that better results can be found by performing a finer tuning of the parameters. Since the Space Pruning Monotonic Search performs very well, we plan to study new

different and powerful heuristics in order to fix the variables. We also plan to apply a multi-objective optimization strategy for the problem, where the constraints are treated as conflicting objective functions; in this way we aim to provide to biologists a better way to analyze the computational results from a biological perspective.

# 5
# Conclusions

Optimization problems are ubiquitous in everyday life and are present in different forms. In particular, in many areas of biomedicine, optimization has become an indispensable tool: drug design and discovery, disease diagnosis and treatment, protein modeling, are just few examples of biomedical issues that benefit from *in silico* design. In fact, computer-aided approaches have an important impact on the reduction of experimental time and cost.
In this thesis, we introduced and discussed two classes of problems that involve the design of genomic sequences: *String Selection and Comparison* (in Chapter 3), and *Probe Design* (in Chapter 4).

**String Selection and Comparison** The first class deals with the identification of common regions within genomic samples. Specifically, similarities and differences among DNA and protein sequences provide relevant information on their function: high similarity usually involves a structural or functional affinity; such insight is used, for instance, to locate genes associated with genetic diseases. Such class of methods addressing the recognition of similar characteristics or differences within biological sequences comprises several problems.
In this thesis, we focused our attention on the *Closest String Problem*, which consists in finding a string with minimum Hamming distance from the sequences of a given finite input set. To overcome the NP-hardness of the problem, we introduced two new approaches: the Ant-CSP algorithm, based on the Ant-Colony Optimization metaheuristic [FP10], and a Simulated Annealing approach, that exploits a new heuristic strategy which allows to find a promising starting point for solving the problem [PCP11].
The Ant-Colony Optimization metaheuristic takes inspiration from real ant colonies, where the behaviour of each single ant is directed to the survival

of the whole colony. In particular, artificial ants are modeled on the foraging behaviour of real ants: when a new food source is found, ants search the shortest and easiest way to return to nest; ants deposit on the ground a chemical substance called pheromone that influences and guides the other ants of the colony in locating the shortest path to reach the food source and coming back to the nest. This approach is very effective in solving NP-hard problems that can be represented as a graph whose characteristics change over time, concurrently with the optimization process. Due to its features, the *Closest String Problem* is suitable for the Ant-Colony Optimization approach. The experimental results conducted show that such method is able to compute good solutions, regardless the number and the length of input strings.

The second algorithm presented for the *Closest String Problem* introduces a new heuristic, called *greedy walk*: as the name states, it identifies a walk on the solution space, that is greedy because the choice of each component is local and myopic. We proved the effectiveness of such heuristic by combining it with a Simulated Annealing algorithm, and we compared our approach with the mathematical formulations and heuristic methods presented in literature for the *Closest String Problem*. Experimental results show that the combination of our heuristic with the Simulated Annealing algorithm computes almost always better solutions than the other heuristic approaches, and has comparable results with exact methods.

Despite the large number of methods presented in literature for the *Closest String Problem*, it is our opinion there is still a lot of work to investigate its computational issues. At this stage, the next step will be a deeper analysis of the heuristic presented in this work, and the investigation of new algorithms and heuristic to obtain not only the improvement of the quality of the solutions, but also an improvement of the performances and convergence speed, since they greatly affect the experimentation costs and time. In such context, the combination of heuristic and exact methodologies might be a successful strategy. On the other hand, we plan to extend our approaches to the *Closest Substring Problem*, due to its biological and computational relevance. Such goal requires an accurate tuning of the proposed algorithms, and a meticulous performance analysis of our approaches for this special case of the *Closest String Problem*, due to its computational intractability.

**Probe Design and Selection** The second class of problems addressed in this thesis involved the design of hybridization probes, used to identify specific targets in biological samples. A probe is a short sequence of DNA or RNA material, labelled with chemical and enzymatical techniques, used to find a specific target sequence on a DNA molecule by performing hybridization experiments. Such experiments allow one to determine whether the probe is present in a DNA solution, i.e. the probe binds with the target. Hybridization is a commonly used technique in disease diagnosis, DNA sequencing, gene expression profiling, drug discovery and development. The selection of "appropriate" probes for hybridization is a difficult task: in particular, selecting unique probes, i.e. probes hybridizing to only one target, is often impracticable. In this thesis we studied the *Non-Unique Probe Selection Problem*, where the selection of a minimum set of non-unique probe is addressed. Due to the genomic relevance of such problem, several approaches have been developed in literature. We proposed two new heuristic approaches for the *Non-Unique Probe Selection Problem*.

The first one is a canonical Monte Carlo algorithm, that implements a heuristic reduction phase to discard "sub-optimal" probes from the final design. Starting from the results obtained by this method, a new combinatorial optimization approach, called Space Pruning Monotonic Search is proposed [POP11b]. This new hybrid method combines the quality guarantee provided by an exact solver, with the exploring ability of an effective and efficient heuristic procedure. To assess the robustness of such methods, we conducted an extensive set of experiments, both on artificial and real data sets. The experimental results have been compared with the state-of-the-art methods: it results that both the Monte Carlo and the Space Pruning Monotonic Search are promising methods and, in particular, the Space Pruning Monotonic Search clearly outperforms the existing approaches for the problem.

Despite the very encouraging results found by our method, we are conscious of being only at the beginning of our research on this class of problems. First, we would like to better explore the combination of exact and heuristic methods for this class of problems, due to the computational effectiveness of hybrid approaches. Moreover, we would like to focus our future research on the target-group separation criteria, since they have not yet been

deeply investigated. From a biological point of view, it could be interesting to extend the existing approaches for the case of higher separation or coverage thresholds; additionally, a further exploration of the biological meaning and features of the datasets used for the experiments might lead to a more effective design.

Moreover, new challenging problems in medicine and biology, such as the analysis of signaling pathways to study drug effects, the identification of Single Nucleotide Polymorphism interactions to detect the risk of certain diseases, the optimization of treatment and drug delivery, represent a new frontier for optimization. In fact, *in silico* approaches play a major role in such contexts, and the necessity of effective optimization methods leads to a new focus on these subjects.

# Bibliography


[ACLS09] B. Addis, A. Cassioli, M. Locatelli, and F. Schoen. A global optimization method for the design of space trajectories. *Computational Optimization and Applications*, pages 1–18, 2009.

[AP03] D.G. Albertson and D. Pinkel. Genomic microarrays in human genetic disease and cancer. *Human molecular genetics*, 12(Review Issue 2):R145, 2003.

[BBH99] RSK Barnes, R.S.K. Barnes, and RN Hughes. *An introduction to marine ecology*. Wiley-Blackwell, 1999.

[BBM08] R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*, volume 45. Springer Verlag, 2008.

[BC96] JE Beasley and P.C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404, 1996.

[BCDV+01] J. Borneman, M. Chrobak, G. Della Vedova, A. Figueroa, and T. Jiang. Probe selection algorithms with applications in the analysis of microbial communities. *Bioinformatics*, 17(Suppl 1):S39, 2001.

[BD04] C. Blum and M. Dorigo. Deception in ant colony optimization. *Ant Colony, Optimization and Swarm Intelligence*, pages 118–129, 2004.

[BDLRP97] A. Ben-Dor, G. Lancia, R. Ravi, and J. Perone. Banishing bias from consensus sequences. In *Combinatorial Pattern Matching*, pages 247–261. Springer, 1997.

[BDT99] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.

[BGH90] LB Booker, DE Goldberg, and JH Holland. Classifier systems and genetic algorithms. *Machine learning: paradigms and methods table of contents*, pages 235–282, 1990.

[BM10] M. Babaie and S.R. Mousavi. A memetic algorithm for closest string problem and farthest string problem. In *Electrical Engineering (ICEE), 2010 18th Iranian Conference on*, pages 570–575. IEEE, 2010.

[Bou10] C.A. Boucher. *Combinatorial and Probabilistic Approaches to Motif Recognition*. PhD thesis, University of Waterloo, 2010.

[BT02] J. Buhler and M. Tompa. Finding motifs using random projections. *Journal of computational biology*, 9(2):225–242, 2002.

[BZJ+03] Z. Bozdech, J. Zhu, M.P. Joachimiak, F.E. Cohen, B. Pulliam, J.L. DeRisi, et al. Expression profiling of the schizont and trophozoite stages of Plasmodium falciparum with a long-oligonucleotide microarray. *Genome Biol*, 4(2):R9, 2003.

[CGF+05] Y. Charbonnier, B. Gettler, P. François, M. Bento, A. Renzoni, P. Vaudaux, W. Schlegel, and J. Schrenzel. A generic approach for the design of whole-genome oligoarrays, validated for genomotyping, deletion mapping and gene expression analysis on Staphylococcus aureus. *BMC genomics*, 6(1):95, 2005.

[CHMS04] H.H. Chou, A.P. Hsia, D.L. Mooney, and P.S. Schnable. Picky: oligo microarray design for large genomes. *Bioinformatics*, 2004.

[CMN04] D. Cazalis, T. Milledge, and G. Narasimhan. Probe selection problem: Structure and algorithms. In *Proc. 8th Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004)*, pages 124–129. Citeseer, 2004.

[CMW11] Z.Z. Chen, B. Ma, and L. Wang. A three-string approach to the closest string problem. *Journal of Computer and System Sciences*, 2011.

[CWB11] M. Chimani, M. Woste, and S. Böcker. A closer look at the closest string and closest substring problem. In *Proceedings of the 13th workshop on algorithm engineering and experiments (ALENEX)*, pages 13–24, 2011.

[DAGP90] J.L. Deneubourg, S. Aron, S. Goss, and JM Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior*, 3(2):159–168, 1990.

[DB05] M. Dorigo and C. Blum. Ant colony optimization theory: A survey. *Theoretical computer science*, 344(2-3):243–278, 2005.

[DCG99] M. Dorigo, G.D. Caro, and L.M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(2):137–172, 1999.

[DG93] K. Deb and D.E. Goldberg. Analyzing deception in trap functions. *Foundations of genetic algorithms*, 2:93–108, 1993.

[DG99] C. Debouck and P.N. Goodfellow. DNA microarrays in drug discovery and development. *nature genetics*, 21(1 suppl):48–50, 1999.

[DH06] D.Z. Du and F.K. Hwang. Pooling designs: Group testing in molecular biology, 2006.

[Dor92] M. Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[DS02] M. Dorigo and T. Stutzle. The ant colony optimization metaheuristic: Algorithms, applications and advances. *Handbook of Metaheuristics*, 57:251–285, 2002.

[dST93] P.S. de Souza and S.N. Talukdar. Asynchronous organizations for multi-algorithm problems. In *Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice*, page 293. ACM, 1993.

[DTMW08] P. Deng, M. Thai, Q. Ma, and W. Wu. Efficient non-unique probes selection algorithms for DNA microarray. *BMC genomics*, 9(Suppl 1):S22, 2008.

[EB00] C.B. Epstein and R.A. Butow. Microarray technology–enhanced versatility, persistent challenge. *Current opinion in biotechnology*, 11(1):36–41, 2000.

[ELD03] S.J. Emrich, M. Lowe, and A.L. Delcher. PROBEmer: a web-based software tool for selecting optimal DNA oligos. *Nucleic Acids Research*, 31(13):3746, 2003.

[FBJ03] A. Figueroa, J. Borneman, and T. Jiang. Clustering binary fingerprint vectors with missing values for dna array data analysis. In *Bioinformatics Conference, 2003. CSB 2003. Proceedings of the 2003 IEEE*, pages 38–47. IEEE, 2003.

[FGN02] M. Fellows, J. Gramm, and R. Niedermeier. On the parameterized intractability of closest substring and related problems. *STACS 2002*, pages 734–734, 2002.

[FL97] M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.

[For95] W. Forster. Handbook of global optimization: Nonconvex optimization and its applications, chapter homotopy methods, 1995.

[FP10] S. Faro and E. Pappalardo. Ant-CSP: An ant colony optimization algorithm for the closest string problem. *SOFSEM 2010: Theory and Practice of Computer Science*, pages 370–381, 2010.

[GGWN09] L.S. Ghoraie, R. Gras, L. Wang, and A. Ngom. Bayesian Optimization Algorithm for the Non-unique Oligonucleotide Probe Selection Problem. In *Proceedings of the 4th IAPR International Conference on Pattern Recognition in Bioinformatics*, pages 365–376. Springer, 2009.

[GGWN10] L.S. Ghoraie, R. Gras, L. Wang, and A. Ngom. Optimal decoding and minimal length for the non-unique oligonucleotide probe selection problem. *Neurocomputing*, 2010.

[GH88] D.E. Goldberg and J.H. Holland. Genetic Algorithms and Machine Learning. *Machine Learning*, 3(2):95–99, 1988.

[GJL99] L. Gasieniec, J. Jansson, and A. Lingas. Efficient approximation algorithms for the Hamming center problem. *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 905–906, 1999.

[GL98] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Pub, 1998.

[GLS09] A. Grosso, M. Locatelli, and F. Schoen. Solving molecular distance geometry problems by global optimization algorithms. *Computational Optimization and Applications*, 43(1):23–37, 2009.

[GLSW07] L. Gasieniec, C.Y. Li, P. Sant, and P.W.H. Wong. Randomized probe selection algorithm for microarray design. *Journal of theoretical biology*, 248(3):512–521, 2007.

[GMPV08] F.C. Gomes, C.N. Meneses, P.M. Pardalos, and G.V.R. Viana. A parallel multistart algorithm for the closest string problem. *Computers & Operations Research*, 35(11):3636–3643, 2008.

[GNR01] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for Closest String and related problems. *Proceedings of the 12th International Symposium on Algorithms and Computation*, pages 441–453, 2001.

[GNR03] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-Parameter Algorithms for Closest String and Related Problems. *Algorithmica*, 37(1):25–42, 2003.

[Gol87] D.E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. *Genetic algorithms and simulated annealing*, 74:88, 1987.

[GS85] RL Graham and N. Sloane. On the covering radius of codes. *Information Theory, IEEE Transactions on*, 31(3):385–401, 1985.

[GS04] P.M.K. Gordon and C.W. Sensen. Osprey: a comprehensive tool employing novel methods for the design of oligonucleotides for DNA sequencing and microarrays. *Nucleic acids research*, 32(17):e133, 2004.

[Haj88] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of operations research*, pages 311–329, 1988.

[Ham50] R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[HHS90] G.Z. Hertz, G.W. Hartzell, and G.D. Stormo. Identification of consensus patterns in unaligned dna sequences known to be functionally related. *Computer applications in the biosciences: CABIOS*, 6(2):81, 1990.

[HJJ03] D. Henderson, S. Jacobson, and A. Johnson. The theory and practice of simulated annealing. *Handbook of metaheuristics*, pages 287–319, 2003.

[Hoc97] D.S. Hochba. Approximation Algorithms for NP-Hard Problems. *ACM SIGACT News*, 28(2):40–52, 1997.

[Hol92] J.H. Holland. *Adaptation in natural and artificial systems*. MIT press Cambridge, MA, 1992.

[HPM+99] R. Herwig, A.J. Poustka, C. Muller, C. Bull, H. Lehrach, and J. O'Brien. Large-scale clustering of cDNA-fingerprinting data. *Genome research*, 9(11):1093, 1999.

[HSL+00] E. Hartuv, A.O. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNA fingerprints. *Genomics*, 66(3):249–256, 2000.

[HSR+08] N. Homer, S. Szelinger, M. Redman, D. Duggan, W. Tembe, J. Muehling, J.V. Pearson, D.A. Stephan, S.F. Nelson, and D.W. Craig. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genet*, 4(8):e1000167, 2008.

[HSS+00] R. Herwig, A.O. Schmitt, M. Steinfath, J. O'Brien, H. Seidel, S. Meier-Ewert, H. Lehrach, and U. Radelof. Information theoretical probe selection for hybridisation experiments. *Bioinformatics*, 16(10):890, 2000.

[HT94] R. Horst and H. Tuy. Global optimization: deterministic approaches. *Journal of the Operational Research Society*, 45(5):595–596, 1994.

[HW04] E.R. Hansen and G.W. Walster. *Global optimization using interval analysis*, volume 264. CRC, 2004.

[Jul09] B.A. Julstrom. A data-based coding of candidate strings in the closest string problem. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2053–2058. ACM, 2009.

[Kel02] E.F. Keller. *The century of the gene*. Harvard Univ Pr, 2002.

[KGV83] S. Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[KK10] T. Kelsey and L. Kotthoff. The exact closest string problem as a constraint satisfaction problem. *Arxiv preprint arXiv:1005.0089*, 2010.

[KL08] A. Koscianski and MA Luersen. Globalization and parallelization of nelder-mead and powell optimization methods. *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, pages 93–98, 2008.

[KRS+04] G.W. Klau, S. Rahmann, A. Schliep, M. Vingron, and K. Reinert. Optimal robust non-unique probe selection using integer linear programming. *Bioinformatics*, 20(suppl 1):i186, 2004.

[KRS+07] G.W. Klau, S. Rahmann, A. Schliep, M. Vingron, and K. Reinert. Integer linear programming approaches for non-unique probe selection. *Discrete Applied Mathematics*, 155(6-7):840–856, 2007.

[KS02] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics*, 18(10):1340, 2002.

[Lan04] J.K. Lanctot. *Some String Problems in Computational Biology*. PhD thesis, University of Waterloo, 2004.

[LDB+96] DJ Lockhart, H. Dong, MC Byrne, MT Follettie, MV Gallo, MS Chee, C. Mittmann, M. and, M. Kobayashi, H. Horton,

et al. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature biotechnology*, 14(13):1675, 1996.

[Lev66] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[LG85] A.V. Levy and S. Gomez. The tunneling method applied to global optimization. In *Numerical optimization 1984: proceedings of the SIAM Conference on Numerical Optimization, Boulder, Colorado, June 12-14, 1984*, volume 1984, page 213. Siam, 1985.

[LHHW08] X. Liu, M. Holger, Z. Hao, and G. Wu. A compounded genetic and simulated annealing algorithm for the closest string problem. In *Bioinformatics and Biomedical Engineering, 2008. ICBBE 2008. The 2nd International Conference on*, pages 702–705. IEEE, 2008.

[LHS05] X. Liu, H. He, and O. Sỳkora. Parallel genetic algorithm and parallel simulated annealing algorithm for the closest string problem. *Advanced Data Mining and Applications*, pages 729–729, 2005.

[LHZ05] X. Li, Z. He, and J. Zhou. Selection of optimal oligonucleotide probes for microarrays using multiple criteria, global alignment and parameter estimation. *Nucleic acids research*, 33(19):6114, 2005.

[LL02] P. Larranaga and J.A. Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Netherlands, 2002.

[LLHM11] X. Liu, S. Liu, Z. Hao, and H. Mauch. Exact algorithm and heuristic for the closest string problem. *Computers & Operations Research*, 2011.

[LLM+99] J.K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 633–642, 1999.

[LM86] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34(1):111–124, 1986.

[LMW99] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 473–482, 1999.

[LMW02] M. Li, B. Ma, and L. Wang. On the closest string and substring problems. *Journal of the ACM (JACM)*, 49(2):157–171, 2002.

[M+29] H.J. Muller et al. The gene as the basis of life. In *International Congress of Plant Sciences, Section of Genetics*, pages 897–921, 1929.

[Ma00] B. Ma. A polynomial time approximation scheme for the closest substring problem. In *Combinatorial Pattern Matching*, pages 99–107. Springer, 2000.

[Mar03] R. Martí. Multi-start methods. *Handbook of metaheuristics*, pages 355–368, 2003.

[Mar08] D. Marx. Closest substring problems with small distances. *SIAM Journal on Computing*, 38(4):1382–1410, 2008.

[MH97] N. Mladenovi and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[MLO+04] C.N. Meneses, Z. Lu, C.A.S. Oliveira, P.M. Pardalos, et al. Optimal solutions for the closest-string problem via integer programming. *INFORMS Journal on Computing*, 16(4):419–429, 2004.

[MMH03] H. Mauch, MJ Melzer, and JS Hu. Genetic algorithm approach for the closest string problem. *Bioinformatics Conference, 2003. CSB 2003. Proceedings of the 2003 IEEE*, pages 560–561, 2003.

[Moc97] J. Mockus. Bayesian heuristic approach to discrete and global optimization: Algorithms, visualization, software, and applications, 1997.

[MOP05] C.N. Meneses, C.A.S. Oliveira, and P.M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *Engineering in Medicine and Biology Magazine, IEEE*, 24(3):81–87, 2005.

[Mos89] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[MPR07] C.N. Meneses, P.M. Pardalos, and M.A. Ragle. A new approach to the non-unique probe selection problem. *Annals of biomedical engineering*, 35(4):651–658, 2007.

[MRR+53] N. Metropolis, AE Rosenbluth, MN Rosenbluth, AH Teller, and E. Teller. Perspective on "Equation of state calculations by fast computing machines". *J. Chem. Phys*, 21:1087–1092, 1953.

[MS08] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. In *Research in Computational Molecular Biology*, pages 396–409. Springer, 2008.

[MSG02] R. Mrowka, J. Schuchhardt, and C. Gille. Oligodb–interactive design of oligo DNA for transcription profiling of human genes. *Bioinformatics*, 18(12):1686, 2002.

[MTG93] S.P. Meyn, R.L. Tweedie, and P.W. Glynn. *Markov chains and stochastic stability*. Springer London et al., 1993.

[MU49] N. Metropolis and S. Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[MVF94] M.A. McClure, T.K. Vasi, and W.M. Fitch. Comparative analysis of multiple protein-sequence alignment methods. *Molecular Biology and Evolution*, 11(4):571, 1994.

[Neu90] A. Neumaier. *Interval methods for systems of equations*, volume 37. Cambridge Univ Pr, 1990.

[Nor05] E.K. Nordberg. YODA: selecting signature oligonucleotides. *Bioinformatics*, 21(8):1365, 2005.

[NRWG10] A. Ngom, L. Rueda, L. Wang, and R. Gras. Selection based heuristics for the non-unique oligonucleotide probe selection problem in microarray design. *Pattern Recognition Letters*, 31(14):2113–2125, 2010.

[NWK03] H.B. Nielsen, R. Wernersson, and S. Knudsen. Design of oligonucleotides for microarrays and perspectives for design of multi-transcriptome arrays. *Nucleic acids research*, 31(13):3491, 2003.

[PCP11] E. Pappalardo, D. Cantone, and P. Pardalos. A combined greedy-walk heuristic and simulated annealing approach for the closest string problem. *(Submitted)*, 2011.

[Pel05] M. Pelikan. Bayesian Optimization Algorithm. *Hierarchical Bayesian Optimization Algorithm*, pages 31–48, 2005.

[Pin02] JD Pintér. Global optimization: software, test problems, and applications. *Nonconvex Optimization and Its Applications*, 62:515–569, 2002.

[PNK94] P. Pudil, J. Novovicová, and J. Kittler. Floating search methods in feature selection. *Pattern recognition letters*, 15(11):1119–1125, 1994.

[POP11a] E. Pappalardo, B. Ozkok, and P. Pardalos. Combinatorial optimization algorithms for probe design and selection problems. *(Submitted)*, 2011.

[POP11b] E. Pappalardo, B. Ozkok, and P. Pardalos. Space pruning monotonic search for the non-unique probe selection problem. *(Submitted)*, 2011.

[PR05] J. Puchinger and G. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. *Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach*, pages 113–124, 2005.

[PRP03] A. Price, S. Ramabhadran, and P.A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, 19(suppl 2):ii149, 2003.

[PS98] C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Dover Pubns, 1998.

[PS00] P.A. Pevzner and S.H. Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, volume 8, pages 269–278. Citeseer, 2000.

[PSdC94] M. Pogu and JE Souza de Cursi. Global optimization by random perturbation of the gradient method with a fixed parameter. *Journal of Global Optimization*, 5(2):159–180, 1994.

[PSP11] E. Pappalardo, G. Stracquadanio, and P. Pardalos. Optimization methods for groundwater supply problems. *(Submitted)*, 2011.

[PSSW06] R.P. Pawlowski, J.N. Shadid, J.P. Simonis, and H.F. Walker. Globalization techniques for newton-krylov methods and applications to the fully coupled solution of the navier-stokes equations. *SIAM Review*, 48(4):700–721, 2006.

[Rag07] M.A. Ragle. *Computational methods for the design and selection of hybridization probes*. PhD thesis, D., UNIVERSITY OF FLORIDA, 2007.

[Rah03] S. Rahmann. Fast large scale oligonucleotide selection using the longest common factor approach. *International Journal Of Bioinformatics And Computational Biology*, 1(2):343–362, 2003.

[RCD+04] N. Reymond, H. Charles, L. Duret, F. Calevro, G. Beslon, and J.M. Fayard. ROSO: optimizing oligonucleotide probes for microarrays. *Bioinformatics*, 20(2):271, 2004.

[RG02] S. Rash and D. Gusfield. String barcoding: Uncovering optimal virus signatures. In *Proceedings of the sixth annual international conference on Computational biology*, pages 254–261. ACM, 2002.

[RHMP05] S. Rimour, D. Hill, C. Militon, and P. Peyret. GoArrays: highly dynamic and efficient microarray probe design. *Bioinformatics*, 21(7):1094, 2005.

[RHZ02] J.M. Rouillard, C.J. Herbert, and M. Zuker. OligoArray: genome-scale oligonucleotide design for microarrays. *Bioinformatics*, 18(3):486–487, 2002.

[RM04] V. Ramos and J.J. Merelo. Self-organized stigmergic document maps: Environment as a mechanism for context learning. *Arxiv preprint cs/0412075*, 2004.

[Rom92] S. Roman. *Coding and information theory*, volume 134. Springer, 1992.

[RS00] S. Rozen and H. Skaletsky. Primer3 on the WWW for general users and for biologist programmers. *Methods Mol Biol*, 132(3):365–386, 2000.

[RSP07] M.A. Ragle, J.C. Smith, and P.M. Pardalos. An optimal cutting-plane algorithm for solving the non-unique probe selection problem. *Annals of biomedical engineering*, 35(11):2023–2030, 2007.

[RSV91] F. Romeo and A. Sangiovanni-Vincentelli. A theoretical framework for simulated annealing. *Algorithmica*, 6(1):302–345, 1991.

[SD02] T. Stuetzle and M. Dorigo. A short convergence proof for a class of ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 6(4):358–365, 2002.

[SH89] G.D. Stormo and G.W. Hartzell. Identifying protein-binding sites from unaligned dna fragments. *Proceedings of the National Academy of Sciences*, 86(4):1183, 1989.

[SL03] WK Sung and WH Lee. Fast and accurate probe selection algorithm for large genomes. In *Proceedings/IEEE Computer Society Bioinformatics Conference. IEEE Computer Society Bioinformatics Conference*, volume 2, page 65, 2003.

[SNC77] F. Sanger, S. Nicklen, and A.R. Coulson. Dna sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463, 1977.

[SPP11] G. Stracquadanio, E. Pappalardo, and P. Pardalos. A mesh adaptive basin hopping method for the design of circular antenna arrays. *(Submitted)*, 2011.

[Ste93] D.E. Stevenson. Science, computational science, and computer science: at a crossroads. In *Proceedings of the 1993 ACM conference on Computer science*, pages 7–14. ACM, 1993.

[STR03] A. Schliep, DC Torney, and S. Rahmann. Group testing with dna chips: generating designs and decoding experiments. In *Proc IEEE Comput Soc Bioinform Conf*, volume 2, pages 84–91, 2003.

[Tan11] S. Tanaka. A heuristic algorithm based on lagrangian relaxation for the closest string problem. *Computers & Operations Research*, 2011.

[TNK+03] N. Tolstrup, P.S. Nielsen, J.G. Kolberg, A.M. Frankel, H. Vissing, and S. Kauppinen. OligoDesign: optimal design of LNA (locked nucleic acid) oligonucleotide capture probes for gene expression profiling. *Nucleic acids research*, 31(13):3758, 2003.

[TZ09] M.T. Thai and T. Znati. On the complexity and approximation of non-unique probe selection using d-disjunct matrix. *Journal of Combinatorial Optimization*, 17(1):45–53, 2009.

[VOR99] S. Voss, I.H. Osman, and C. Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Kluwer Academic Publishers, 1999.

[WC53] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids. *Nature*, 171(4356):737–738, 1953.

[WN07] L. Wang and A. Ngom. A model-based approach to the non-unique oligonucleotide probe selection problem. In *Bio-Inspired Models of Network, Information and Computing Systems, 2007. Bionetics 2007. 2nd*, pages 209–215. IEEE, 2007.

[WNGR08] L. Wang, A. Ngom, R. Gras, and L. Rueda. An evolutionary approach to the non-unique oligonucleotide probe selection problem. *Transactions on Computational Systems Biology X*, pages 143–162, 2008.

[WNR08] L. Wang, A. Ngom, and L. Rueda. Sequential Forward Selection Approach to the Non-unique Oligonucleotide Probe Selection

Problem. *Pattern Recognition in Bioinformatics*, pages 262–275, 2008.

[WS03] X. Wang and B. Seed. Selection of oligonucleotide probes for protein coding sequences. *Bioinformatics*, 19(7):796, 2003.

[WZ09] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. *Frontiers in Algorithmics*, pages 261–270, 2009.

[ZE81] S.H. Zanakis and J.R. Evans. Heuristic "optimization": why, when, and how to use it. *Interfaces*, pages 84–91, 1981.

[Zhi91] A.A. Zhigljavsky. *Theory of global random search*. Kluwer Academic Publishers, 1991.

[ZZ95] Q. Zheng and D. Zhuang. Integral global minimization: algorithms, implementations and numerical tests. *Journal of Global Optimization*, 7(4):421–454, 1995.