M.Sc. Engg. Thesis

Practically Efficient Algorithms for Minimum String Cover and Minimum Common String Partition problem

by
S. M. Ferdous

Submitted to

Department of Computer Science and Engineering

in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

(BUET)

Dhaka 1000

March 2014

The thesis titled '**Practically Efficient Algorithms for Minimum String Cover and Minimum Common String Partition problem**', submitted by S. M. Ferdous, Roll No. **0411052014P**, Session April 2011, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on March 19, 2014.

Board of Examiners

Candidate's Declaration

This is hereby declared that the work titled "Practically Efficient Algorithms for Minimum String Cover and Minimum Common String Partition problem" is the outcome of research carried out by me under the supervision of Dr. M. Sohel Rahman, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

S. M. Ferdous

S. M. Ferdous
Candidate

Acknowledgements

I express my heart-felt gratitude to my supervisor, Dr. M. Sohel Rahman for his constant supervision of this work. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success. Throughout the thesis work, his constant supervision and guidance helped me to learn a lot of new things those will certainly enrich my future career.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank , Dr. M. Kaykobad and Dr. Md. Yusuf Sarwar Uddin and specially the external member Dr. Mohammad Shafiul Alam.

I would also want to thank Anindya Das, now a research assistant at Iowa State University for his ideas in some part of the thesis.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

Abstract

Covering and *Partitioning* problems are very important problems in computer science. Both of them deal with combinatorial structure of the instances. In covering problem the question is whether a certain combinatorial structure covers another. On the other hand the partition problem asks for partitioning a certain combinatorial structure to smaller structures maintaining some constraints. A prominent example of covering problem is Set Cover problem while an important partitioning problem is Integer Partition problem. In these thesis, we are interested in two string related covering and partitioning problem, namely, Minimum String Cover (MSC) problem and Minimum Common String Partition (MSCP) problem. The first one is a covering problem and the second one is the partitioning problem. MSC problems has its application in formal language theory, protein folding and in text compression while MCSP has its direct application in genome comparison.

In this thesis, we are interested in designing efficient and practical algorithms for solving MSC and MCSP problems. As both of them share the NP-hard complexity status and are combinatorial optimization problem, we have applied variants of Ant Colony Optimization (ACO) techniques to solve them. For MCSP we also developed the Mixed Integer Linear Programming formulation. An extensive experiment is carried that compares our result with the state of the art algorithms for these two problems.

Contents

List of Figures

List of Tables

List of Algorithms

Dedicated to my father...

Publications out of this thesis work

1. Ferdous, S., Das, A., Rahman, M., Rahman, M.: Ant colony optimization approach to solve the minimum string cover problem. In: International Conference on Informatics, Electronics & Vision (ICIEV), IEEE (2012) 741-746

2. Ferdous, S., Rahman, M.: Solving the minimum common string partition problem with the help of ants. In Tan, Y., Shi, Y., Mo, H., eds.: Advances in Swarm Intelligence. Volume 7928 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 306-313

Chapter 1

Introduction

String problems, a set of combinatorial problems, are seminal topics in computer science. Various string problems with different complexity status existed in the literature. In this thesis, we have explored two string problems, namely Minimum String Cover (MSC) and Minimum Common String Partition (MCSP) problems. The first problem is a cover problem on a set of strings. In MSC problem we are given a set of strings called target string. For each of the substring of the target string set, we define a cost function. In MSC, we have to find the minimum cost string set that is a subset of the substring set of the target strings and the target strings can be regenerated by any combination of the chosen string set. On the other hand, the second problem is a partition problem defined on a couple of strings. MCSP problem is the more restricted version of MSC problem, where the partition set must be common between the two strings. In the MCSP, we are given two strings on input with equal frequency on alphabet set, and we wish to partition them into the same collection of substrings, minimizing the number of the substrings in the partition. Both of these problems share the NP-hard complexity staus and are related in their mapping in the underlying graph. Although some theoretical work on these two problems exist in the literature, there are very few practical analysis of these problems exist in the domain. In this research work, we have developed some practical and efficient algorithms to solve these problems. Our algorithms have been tested extensively on different benchmark test.

This chapter will serve as an introduction of the topics and thesis . we will discuss about Minimum String Cover (MSC) and Minimum Common String Partition (MCSP) problems and motivation of these two problems. Also we will discuss about our scope of this work and outcome of this thesis. Finally this chapter concludes with a overview of the rest of the chapters.

1.1 Practical applications of MSCP

A string cover C of a set of strings S is a set of substrings from S such that every string in S can be written as a concatenation of the strings in C. Given costs assigned to each substring from S, the Minimum String Cover (MSC) problem asks for a cover of minimum total cost. The complexity of the problem is NP-hard. Although the problem is very hard to solve it has several applications in different domain.

1.1.1 Protein Folding

In [16], Bodlaender et al. described an application for this problem in the context of protein folding. (They referred to the problem as the Dictionary Generation problem, and considered its unweighted variant under the parameterized complexity framework.) Protein folding is the problem of determining the folding structure of proteins using their amino-acid sequential description. This problem is extremely important, since most of the functionality of a protein is determined by its folding structure, and because current biological methods for extracting the sequential description of a given protein exceed by far the methods for extracting the folding structure of the protein. In [16], it is argued that since all known approaches for protein folding are NP-hard, a possible heuristic for this problem is to break the protein sequence into small segments, small enough for allowing efficient folding computation. This heuristic is justified by the fact that many proteins seem to be composed of relatively small regions which fold independently of other regions. The theory of exon shuffling proposes that all proteins are concatenations of such regions, where the regions are drawn from a common ancestral dictionary [32].

1.1.2 Formal Language Theory

MSC can also model interesting computational issues which arise in formal language theory, and in particular, in the area of combinatorics of words. Our notion of cover actually corresponds to the notion of combinatorial rank, an important parameter of a set of words . Neraud [65] studied the problem of determining whether a given set of words is elementary, where a set of strings is said to be elementary if it does not have a cover of size strictly less than its own.

1.1.3 Text Compression

Another important application of the problem can be found in data compression. In data compression, the task is encoding information in fewer bits. If the bits get lost in the compression process it is termed as lossy compression. On the other hand if the original information can be fully decoded from the encoded compressed information, the compression technique is called lossless compression. It can easily be verified that, MSC can be used to generate a simple loss less compression algorithm. Suppose we have a text with n lines of strings. If we can find the cover of the n strings, the cover set can be used to index the text file. By this the original file can be compressed. An example is illustrated as follows:

Example 1.1. *Suppose we have text, $T = \{$ "abcad", "adabcdb", "dbadabbc", "abcabbcdb", "abcdb", "abbcad", "dbaddbabbc", "abcadabb"$\}$. One of the cover, $C = \{$ "abc", "db", "ad", "abbc"$\}$. After establishing the cover, we can rewrite the original text by the index of the cover string. The text, T can be encoded as: $\{02, 201, 123, 031\}$, where the numbers are the index of the cover set C starting from 0. It can be easily verified that the decoding is lossless.*

1.2 Practical application of MCSP

In the Minimum Common String Partition (MCSP) problem, two strings X, Y of length n, that contain the same symbols equally many times, shall be partitioned each into k segments called blocks, so that the blocks in X and Y constitute the same multiset of substrings. This problem is also proved to be NP-hard.

1.2.1 Measure of Evolutionary Distance

Evolutionary distance between two DNA strings of different species is a measure of their similarities. Computing evolutionary distance is an important topic in

Comparative Genomics [36]. Like Sorting by reversals [7] and Sorting by transpositions [6] MCSP is a measure of evolutionary distance. Given two DNA strings, MCSP answers the possibilities of re-arrangement of one DNA string to another [22]. In MCSP, the transformation criteria is the Common String Partition(CSP).

Example 1.2. *Suppose we have two species. The DNA string of the first species is, "gtacggtcaagc". And the DNA string of the second species is, "aagcgtcagtac". One of the minimum CSP set is {"gtac", "ggtc" and "aagc"} and CSP size is 3.*

1.2.2 Ortholog Assignment of Genes

MCSP is also important in ortholog assignment. In [19], the authors present a new approach to ortholog assignment that takes into account both sequence similarity and evolutionary events at a genomic level. In that approach, first, the problem is formulated as that of computing the signed reversal distance with duplicates between the two genomes of interest. Then, the problem is decomposed into two optimization problems, namely minimum common string partition and maximum cycle decomposition problem. Thus MCSP plays an integral part in computing ortholog assignment of genes.

1.3 Scope of the thesis

In this thesis, we are interested in developing practical and efficient algorithms for solving MSC and MCSP problems. We are interested in Metaheuristic approaches to solve the problems. To the best of our knowledge, there exists no attempt to solve these problems with Meta-heuristic approaches. Particularly we are interested in nature inspired algorithms. As the two problems are discrete combinatorial optimization problems, the natural choice is Ant Colony Optimization (ACO).

1.4 Contribution of the thesis

The objective of this research is to develop efficient and practical algorithms for solving these two problems. We have contributed in mapping the problems into graphs. Upon these graphs ACO is implemented with various static and dynamic heuristics. For MCSP problem, we have also developed a mixed integer programming formulation. It was solved by branch and cut method. In this section we will briefly outline the contribution of the thesis.

1.4.1 Graph mapping

MSC and MCSP are NP-hard problems. So, no exact polynomial algorithms exist (unless it is proved that P=NP). We have to rely on approximation algorithm or metaheuristic approaches. As we have chosen ACO to solve these two problems, we first need to map the problems into graphs. In our work for MCSP we have developed *common substring graph* on two strings. The mapping of the graph from the input string can be done in polynomial time. For MSC problem, we have borrowed and modified *substring graph* from [18].

1.4.2 Mixed Integer Programming formulation of MCSP

There are two Mixed Integer Programming (MIP) formulation of MSC problem in the literature [18, 48]. But no MIP formulation is found for MCSP problem. In our work, we have developed an efficient MIP formulation of MCSP problem.

1.4.3 Developing dynamic heuristic

Heuristics play important roles in the performance in ACO algorithms. For MCSP problem, we have developed some heuristics that effect solution quality positively.

1.4.4 Developing ACO algorithms

Using the mapping graphs and heuristics developed, we have implemented different ACO algorithms, namely MAX-MIN Ant System (MMAS), Ant Colony System (ACS) and Hybrid Ant System (HAS) on these two problems. The parameters of the specific algorithms needed to be set to ensure the best results.

1.4.5 Extensive experiment

We conducted experiments for a detailed performance evaluation of our algorithms and compared our algorithms with the best known algorithms available in the literature. The supremacy of our algorithms are justified by the comparative analysis performed based on this experiments.

1.5 Overview of the thesis

The rest of the chapters are organized as follows. Chapter 2 presents the preliminary concepts necessary to comprehend the rest of the thesis. In particular, this chapter briefly introduces MSC and MCSP problems and discusses definitions and notations related to this thesis work, and finally presents a brief description of the Metaheuristic techniques namely ant colony optimization (ACO). Chapter 3 presents related works and the existing algorithms to solve MSC and MCSP problems. Chapter 4 and Chapter 5 contain the algorithms those were developed for solving the two problems. Chapter 6 contains the experimental results of our schemes and a comparative study with the existing state of the art algorithms on several performance issues. Chapter 7 draws the conclusion describing the key contributions of this thesis followed by some future research directions related to this topic.

Chapter 2

Preliminaries

In this chapter we define MSC and MCSP problems formally and elaborately. We will also mention some notations and definitions that will be used throughout the thesis. Lastly we will discuss the basics of Ant Colony Optimization and Integer Linear Programming.

2.1 Strings

Traditionally string is an umbrella term for sequences of symbols or characters. To formally approach strings, we need an alphabet typically denoted by Σ and the *grammar* of the *language* or the set of strings that we accept.

Definition 2.1. *String*: A *string* (or word) over Σ is any finite sequence of symbols from Σ [69].

Example 2.1. *If* $\Sigma = \{0, 1\}$, *then 01011 is a string over* Σ.

In this thesis, we only consider finite languages, that is, sets of strings with a finite number of elements, which can be modeled with regular expressions as defined in [53] A regular expression over an alphabet Σ is either one of the following constant symbols:

- the empty set ϕ
- the empty string ϵ

- a single symbol of Σ

or a combination of regular expressions s and t using the following operators:

- the concatenation st
- the alteration $s + t$
- the kleene star s^*

The regular expression corresponding to all possible strings over an alphabet $\Sigma = \{A, B, C, ..., \}$ is often written as Σ^*. Concatenation is an important binary operation on Σ^*. For any two strings s and t in Σ^*, their concatenation is defined as the sequence of symbols in s followed by the sequence of characters in t, and is denoted st. The alteration of two regular expressions merges the languages, that is, all strings represented by s and all strings represented by t. The Kleene star represents all strings that can be constructed by concatenating strings represented by s an arbitrary number of times.

Definition 2.2. *Substring*: A string s is said to be a *substring* or factor of t if there exist (possibly empty) strings u and v such that $t = usv$. We define $C(t)$ to be the set of all substrings of t.

Example 2.2. *If t = "stringcoverandpartiion" then s_1= "cover" and s_2 = "partition" are both substrings of t.*

Definition 2.3. *Prefix and Suffix*: A string s is said to be a *prefix* of t if there exists a string u such that $t = su$. If u is nonempty, s is said to be a proper prefix of t. Symmetrically, a string s is said to be a *suffix* of t if there exists a string u such that $t = us$. If u is nonempty, s is said to be a proper suffix of t. *Suffixes* and *prefixes* are substrings of t.

Example 2.3. *If, t = "stringcoverandpartiion", then s_1 = "stringcover" and s_2= "partition" are two of the prefixes and suffixes respectively.*

2.2 Graphs

The definitions in this section is based on [55].

A directed graph $G(V, E)$ contains a non-empty set V of nodes and a set $E \subseteq V^2$ of edges. An edge $e = (u, v) \in E$ starts at node u and ends at node v. Then, we say that $u \in V$ and $v \in V$ are interconnected or adjacent. Further, both nodes are incident with edge e. E is not necessarily symmetrical, that is, $(u, v) \in E$ does not mean that $(v, u) \in E$ as well. A *weighted directed graph* G is a triple of (V, E, f) with $G(V, E)$ representing a directed graph and a weight function $f : E \to R$.

A path b from node $v \in V$ to node $u \in V$ in a weighted directed graph $G(V, E, f)$ is a sequence of edges $e_1, e_2, \ldots e_n$ for some positive integer n such that edge e_i ends at the same node at which edge ei+1 starts. Furthermore, edge e1 starts at node v and edge en ends at node u. The weighted length of path b is the sum of its edge weights, that is

$$f(b) = \sum_{i=1}^{n} f(e_i) \tag{2.1}$$

If there exists a path in $G(V, E, f)$ from node $v \in V$ to node $u \in V$, we say that u is *reachable* from v. A path starting and ending with the same node is called a *cycle*. A graph without any cycles is called *acyclic*.

2.3 Notations and Definitions

In this section, we present some definitions and notations that are used throughout the thesis.

Definition 2.4. *Related string:* Two strings (X, Y), each of length n, over an alphabet $\sum$ are called *related* if every letter appears the same number of times in each of them.

Example 2.4. *X = "abacbd" and Y = "acbbad", then they are* related. *But if X_1 = "aeacbd" and Y = "acbbad", they are not* related

Definition 2.5. *Block*: A block $B = ([id, i, j])$, $0 \leq i \leq j < n$, of a string S is a data structure having three fields: *id* is an identifier of S and the starting and ending positions of the block in S are represented by i and j, respectively. Naturally, the *length* of a block $[id, i, j]$ is $(j - i + 1)$. We use *substring*$([id, i, j])$ to denote the substring of S induced by the block $[id, i, j]$. Throughout the report we will use 0 and 1 as the identifiers of X(i.e., *id*(X)) and Y(i.e., *id*(Y)) respectively. We use [] to denote an empty block.

Example 2.5. *If we have two strings* $(X, Y) = \{$ *"abcdab", "bcdaba"*$\}$*, then* $[0, 0, 1]$ *and* $[0, 4, 5]$ *both represent the substring "ab" of* X*. In other words,* $substring([0, 0, 1]) = substring([0, 4, 5]) =$ *"ab".*

Two blocks can be intersected or unioned. The intersection of two blocks (with same ids) is a block that contains the common portion of the two.

Definition 2.6. *Intersection of blocks*: Formally, the intersection operation of $B_1{=}[id, i, j]$ and $B_2{=}[id, i', j']$ is defined as follows:

$$B_1 \cap B_2 = \begin{cases} [] & \text{if } i' > j \text{ or } i > j' \\ [id, i', j] & \text{if } i' \leq j \\ [id, i, j'] & \text{else} \end{cases} \tag{2.2}$$

Example 2.6. *If,* $B_1 = [0, 1, 5]$ *and* $B_2 = [0, 3, 6]$*, then* $B_1 \cap B_2 = [0, 3, 5]$*. On the other hand, if* $B_1 = [0, 1, 5]$ *and* $B_2 = [0, 6, 8]$*, then* $B_1 \cap B_2 = []$

Definition 2.7. *Union of blocks*: Union of two blocks (with same ids) is either another block or an ordered (based on the starting position) set of blocks. Without the loss of generality we suppose that, $i \leq i'$ for $B_1{=}[id, i, j]$ and $B_2{=}[id, i', j']$. Then, formally the union operation of B_1 and B_2 is defined as follows:

$$B_1 \cup B_2 = \begin{cases} [id, i, j] & \text{if } j' <= j \\ [id, i, j'] & \text{if } j' > j \text{ or } i' == j + 1 \\ \{B_1, B_2\} & \text{else} \end{cases} \tag{2.3}$$

Example 2.7. *If,* $B_1 = [0, 1, 5]$ *and* $B_2 = [0, 3, 6]$*, then* $B_1 \cup B_2 = [0, 1, 6]$*. On the other hand, if* $B_1 = [0, 1, 5]$ *and* $B_2 = [0, 6, 8]$*, then* $B_1 \cup B_2 = \{[0, 1, 5], [0, 6, 8]\}$

The union rule with an ordered set of blocks, B_{lst} and a block, B' can be defined as follows. We have to find the position where B' can be placed in B_{lst}, i.e., we have to find $B_k \in B_{lst}$ after which B' can be placed. Then, we have to replace the ordered subset $\{B_k, B_{k+1}\}$ with $B_k \cup B' \cup B_{k+1}$.

Example 2.8. *As an example, suppose we have three blocks, namely,* $B_1 = [0, 5, 7]$*,*$B_2 = [0, 11, 12]$ *and* $B_3 = [0, 8, 10]$*. Then* $B_1 \cup B_2 = B'_{lst} = \{[0, 5, 7], [0, 11, 12]\}$*. On the other hand,* $B'_{lst} \cup B_3 = [0, 5, 12]$*, which is basically identical to* $B_1 \cup B_2 \cup B_3$*.*

Two blocks B_1 and B_2 (in the same string or in two different strings) matches if $substring(B_1) = substring(B_2)$. If the two matched blocks are in two different strings then the matched substring is called a common substring of the two strings denoted by *cstring(*B_1, B_2*)*.

Definition 2.8. *span*: Given a list of blocks with same id, the *span* of a block, $B = [id, i, j]$ in the list denoted by, $span(B)$ is the length of the block (also in the list) that contains B and whose length is maximum over all such blocks in the list. Note that a block is assumed to contain itself. More formally, given a list of blocks, $list_b$, $span(B \in list_b) = \max\{\ell \mid \ell = length(B'), B \subseteq B', \forall B' \in list_b\}$.

Example 2.9. *If* $list_b = \{[0,0,0],[0,0,1],[0,0,2],[0,4,5]\}$ *then* $span([0,0,0]) = span([0,0,1]) = span([0,0,2]) = 3$ *where as,* $span([0,4,5]) = 2$*. In other words, span of a block is the maximum length of the super string than contains the substring induced by the block.*

Definition 2.9. *Partition*: A *partition* of a string X is a list of blocks all with $id(X)$ having the following two properties:

(a) Non Overlapping: The blocks must be be disjoint, i.e., no block should overlap with another block. So the intersection of any two blocks must be empty.

(b) Cover: The blocks must cover the whole string.

In other words, a *partition* of a string X is a sequence $P = (B_1, B_2, \ldots, B_m)$ of strings whose concatenation is equal to X, that is $B_1 B_2 \ldots B_m = X$. where B_i's are *blocks*.

Definition 2.10. *Cover*: Given a set of strings S, We say, $C \subset C(S)$ is a Cover of S if each string in S can be written as a concatenation of some strings in C.

Example 2.10. *Let,* $S = \{abc, cab, bc\}$*. So,* $C = \{ab, b, c\} \subset C(S)$ *is a cover of* S.

Definition 2.11. *Substring graph*: *Substring graph* models the factorization of a string. Formally, for a string s of length n can be mapped into a *substring graph*, $G_s(V_s, E_s)$, where,

$$V_s = \{(s,0),(s,1),...,(s,n)\} \tag{2.4}$$

$$E_s = \{(s,p),(s,q) : p < q, (s,p) \in V_s, (s,q) \in V_s\} \tag{2.5}$$

The directed edge $((s,p),(s,q))$ represents the sub- string interval $(s, p, q-1)$, spelling a substring of length $q - p$.

2.4 Ant Colony Optimization

Ant colony optimization (ACO) [25, 27, 28, 31] was introduced by M. Dorigo and colleagues as a novel nature-inspired metaheuristic for the solution of hard combinatorial optimization (CO) problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants which use pheromones as a communication medium. In analogy to the biological example, ACO is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as a distributed, numerical information which the ants use to probabilistically construct solutions to the problem being solved and which the ants adapt during the algorithm's execution to reflect their search experience.

In the real world, ants (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep traveling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food. Over time, however, the pheromone trails start to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over faster, and thus the pheromone density remains high as it is laid on the path possibly faster than it can evaporate. Pheromone evaporation has also the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. Thus, when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.

The original idea comes from observing the exploitation of food resources among ants, in which ants with individually limited cognitive abilities have collectively been able to find the shortest path between a food source and the nest. An illustrative example is presented at Figure 2.1

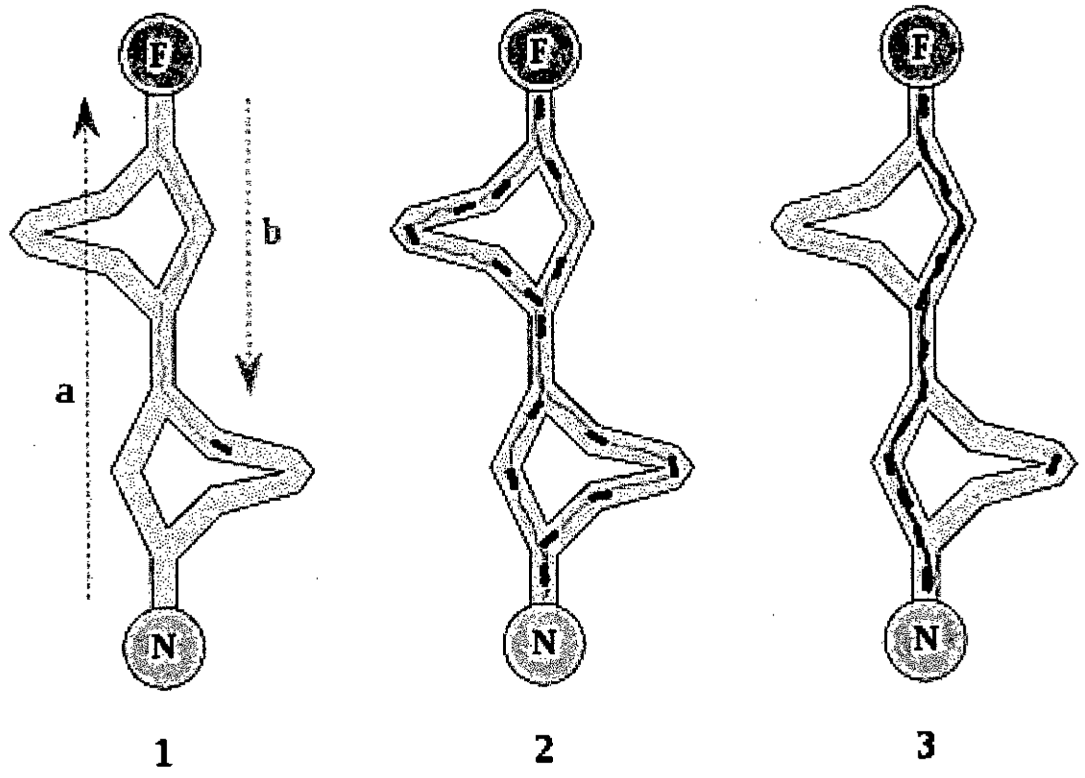

FIGURE 2.1: Ant Colony Optimization

1. The first ant finds the food source (F), via any way (a), then returns to the nest (N), leaving behind a trail pheromone (b).
2. Ants indiscriminately follow four possible ways, but the strengthening of the runway makes it more attractive as the shortest route.
3. Ants take the shortest route, long portions of other ways lose their trail pheromones.

In a series of experiments on a colony of ants with a choice between two unequal length paths leading to a source of food, biologists have observed that ants tended to use the shortest route. A model explaining this behavior is as follows:

1. An ant (called "blitz") runs more or less at random around the colony.
2. If it discovers a food source, it returns more or less directly to the nest, leaving in its path a trail of pheromone.
3. These pheromones are attractive, nearby ants will be inclined to follow, more or less directly, the track.

4. Returning to the colony, these ants will strengthen the route.
5. If two routes are possible to reach the same food source, the shorter one will be, in the same time, traveled by more ants than the long route will.
6. The short route will be increasingly enhanced, and therefore become more attractive.
7. The long route will eventually disappear since pheromones are volatile.
8. Eventually, all the ants have determined and therefore "chose" the shortest route.

2.4.1 Basics of ACO

In general, the ACO approach attempts to solve a combinatorial optimization (CO) problem by iterating the following two steps.. At first, solutions are constructed using a pheromone model, i.e., a parameterized probability distribution over the solution space. Then, the solutions that were constructed in earlier iterations are used to modify the pheromone values in a way that is deemed to bias the search towards the high quality solutions.

2.4.1.1 Ant Based Solutions Construction

As mentioned above, the basic ingredient of an ACO algorithm is a constructive heuristic to probabilistically construct solutions. A constructive heuristic assembles solutions as sequences of solution components taken from a finite set of solution components $C = \{c_1, c_2, ...c_n\}$. A solution construction starts with an empty partial solution $s^p = \emptyset$. Then at each construction step the current partial solution s^p is extended by adding a feasible solution component from the solution space C. The process of constructing solutions can be regarded as a walk (or a path) on the so-called construction graph $G_c = (C, E)$ whose vertices are the solution components C and the set E are the connections (i.e., edges).

2.4.1.2 Heuristic Information

In most ACO algorithms the transition probabilities, i.e., the probabilities for choosing the next solution component, are defined as follows:

$$p(c_i|s^p) = \frac{\tau_i{}^{\alpha} \cdot \eta(c_i)^{\beta}}{\sum_{c_j \in N(s^p)} \tau_j{}^{\alpha} \cdot \eta(c_j)^{\beta}}, \forall c_i \in N(s^p) \tag{2.6}$$

Here, η is a weight function that contains *heuristic information* and α, β are positive parameters whose values determine the relation between the *pheromone information* and the *heuristic information.* The pheromones deployed by the ants are denoted by τ.

2.4.1.3 Pheromone Update

pheromone update consists of two parts. First, a pheromone evaporation, which uniformly decreases all the pheromone values, is performed. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a sub-optimal region. It helps to forget the local optimal solutions and thus favors the exploration of new areas in the search space. Then, one or more solutions from the current or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions. As a prominent example, we describe the following pheromone update rule that was used in Ant System (AS) [27], which was the first ACO algorithm proposed.

$$\tau_i \leftarrow (1-\varepsilon) \times \tau_i + \tau_i \times \sum_{s \in G_{iter} | c_i \in s} F(s) \times \varepsilon, i = 1, 2, ..., n \tag{2.7}$$

Let $W(.)$ is the cost function. Here, G_{iter} is the set of solutions found in the current iteration, $\varepsilon \in (0, 1]$ is a parameter called the *evaporation rate*, and $F : G \to \mathbb{R}^+$ is a function such that $W(s) < W(\acute{s}) \Rightarrow F(s) \geq F(\acute{s}), s \neq \acute{s}, \forall s \in G$. The function $F(.)$ is commonly called the *Fitness Function.*

2.4.2 Variants of ACO

Different ACO algorithms have been proposed in the literature. The original algorithm is known as the Ant System(AS) [24, 26, 29]. The other variants are, Elitist AS [24, 29], ANT-Q [39], Ant Colony System (ACS) [28], MAX-MIN AS [82–84] etc. Below we will discuss some important variants of ACO.

2.4.2.1 Ant System

Ant System is the first ACO algorithm proposed in the literature [24, 26, 29]. Its main characteristic is that, at each iteration, the pheromone values are updated by all the m ants that have built a solution in the iteration itself.

2.4.2.2 MAX-MIN Ant System (MMAS)

The MMAS algorithm is characterized as follows. Firstly, the pheromone values are limited to an interval $[\tau_{MIN}, \tau_{MAX}]$ with $0 < \tau_{MIN} < \tau_{MAX}$. Pheromone trails are initialized to τ_{max} to favor the diversification during the early iterations so that premature convergence is prevented. Explicit limits on the pheromone values ensure that the chance of finding a global optimum never becomes zero. Secondly, in case the algorithm detects that the search is too much confined to a certain area in the search space, a restart is performed. This is done by initializing all the pheromone values again. Thirdly, the pheromone update is always performed with either the iteration-best solution, the restart-best solution (i.e., the best solution found since the last restart was performed), or the best-so-far solution.

2.4.2.3 Ant Colony System (ACS)

The ACS algorithm was introduced to improve over the performance of Ant System (AS). ACS is based on AS with some important differences. Firstly, after each iteration, a pheromone update is done using only the best solution found so far. The pheromone evaporation is only applied to the solution components that are in the best-so-far solution. Secondly, the transition probabilities are defined by a rule that is called *pseudo-random-proportional* rule. With this rule, some construction steps are performed in a deterministic manner, whereas in others the transition probabilities are defined. Let q is a random number and q_0 is a threshold value, then the high level description of *pseudo-random-proportional* is,

$$Next\ component = \begin{cases} best\ available\ component & \text{if } q \leq q_0 \\ draw\ according\ 2.6 & \text{otherwise} \end{cases}$$

Thirdly, during the solution construction the pheromone value of each added solution component is slightly decreased.

2.4.3 Application of ACO

Recently, growing interest has been noticed towards ACO in the scientific community. There are now available several successful implementations of the ACO metaheuristic applied to a number of different discrete combinatorial optimization problems . In [27], the authors distinguished between two classes of applications of ACO: those to static combinatorial optimization problems, and those to the dynamic ones. When the problem is defined and does not change while the problem is being solved is termed as static combinatorial optimization problems. The authors in [27], listed some static combinatorial optimization problems those are successfully solved by different variants of ACO. Some of the problems are, traveling salesperson, Quadratic Assignment, job-shop scheduling, vehicle routing, sequential ordering, graph coloring etc. Dynamic problem is defined as a function of some quantities whose values are set by the dynamics of an underlying system. The problem changes therefore at run time and the optimization algorithm must be capable of adapting online to the changing environment. The authors in [27], listed connection-oriented network routing and connectionless network routing as the examples of dynamic problems those are successfully solved by ACO.

In 2010 a non-exhaustive list of applications of ACO algorithms grouped by problem types are presented in [30]. The authors categorized the problems into different types namely routing, assignment, scheduling, subset machine learning and bioinformatics. In each type they listed the problems those are successfully solved by some variants of ACO. Some current applications of ACO algorithms are listed in Table 2.1.

TABLE 2.1: Some current applications of ACO algorithms (adapted from [30]).

Problem type	*Problem name*	*Authors*	*Year*	*References*
Routing	Traveling salesman	Dorigo et al.	1991, 1996	[26, 29]
		Dorigo and Gambardella	1997	[28]
		Stutzle and Hoos	1997, 2000	[83]
	TSP with time windows	López Ibáñez et al.	2009	[58]
	Sequential ordering	Gambardella and Dorigo	2000	[40]
	Vehicle routing	Gambardella et al.	1999	[41]
		Reimann et al.	2004	[72]
		Fuellerer et al.	2009	[38]
	Multicasting	Hernandez and Blum	2009	[49]
Assignment	Quadratic assignment	Maniezzo	1999	[59]
		Stützle and Hoos	2000	[84]
	Frequency assignment	Maniezzo and Carbonaro	2000	[60]
	Course timetabling	Socha et al.	2002, 2003	[77, 78]
	Graph coloring	Costa and Hertz	1997	[21]
Scheduling	Project scheduling	Merkle et al.	2002	[64]
	Weighted tardiness	den Besten et al.	2000	[11]
		Merkle and Middendorf	2000	[63]
	Flow shop	Stützle	1997	[85]
		Rajendran, Ziegler	2004	[71]
	Open shop	Blum	2005	[12]
	Car sequencing	Solnon	2008	[79]
Subset	Set covering	Lessing et al.	2004	[57]
	l-cardinality trees	Blum and Blesa	2005	[14]
	Maximum clique	Solnon and Fenet	2006	[80]
Machine Learning	Classification rules	Parpinelli et al.	2002	[68]
		Martens et al.	2006	[62]
		Otero et al.	2008	[66]
	Bayesian networks	Campos, Fernàndez-Luna	2002	[23]
	Neural networks	Socha, Blum	2007	[76]

2.4.3.1 String algorithms and ACO

There are not too many string related problems solved by ACO in the literature. In [15], the authors addressed the reconstruction of DNA sequences from DNA fragments by ACO. Several ACO algorithms have been proposed for the longest common subsequence (LCS) problem in [13, 75]. Closest string problem(CSP) is an important problem is sequence analysis. Ant Colony algorithms have been proposed for CSP in [8, 35]

2.5 Mixed Integer Linear Problem (MILP)

A common approach to modeling optimization problems with discrete decisions is to formulate them as mixed integer optimization problems. In this section we will briefly overview the structure of MILP and their applications in discrete optimization problems. This section is based on [51].

Definition 2.12. *Mixed Integer Linear Problem (MILP)*: The problems in which the functions required to represent the objective and constraints are additive, i.e., linear functions. Such is called a *mixed integer linear optimization problem* (MILP). The general form is

$$max \sum_{j\in B} c_j x_j + \sum_{j\in I} c_j x_j + \sum_{j\in C} c_j x_j \tag{2.8}$$

subject to,

$$\sum_{j\in B} a_{ij} x_j + \sum_{j\in I} a_{ij} x_j + \sum_{j\in C} a_{ij} x_j \left\{ \begin{array}{c} \leq \\ = \\ \geq \end{array} \right\} b_i \ \ \forall_i \in M \tag{2.9}$$

$$l_j \leq x_j \leq u_j \ \ \forall j \in N = B \cup I \cup C \tag{2.10}$$

$$x_j \in 0, 1 \ \ \forall j \in B \tag{2.11}$$

$$x_j \in Z \ \ \forall j \in I \tag{2.12}$$

$$x_j \in R \ \ \forall j \in C \tag{2.13}$$

A *solution* to Eqs. (2.8)-(2.13) is a set of values assigned to the variables x_j, $j \in N$. This *solution* set is sometimes referred as feasible region, feasible set, search space, or solution space. The objective is to find a solution that maximizes the weighted sum Eq. (2.8), where the coefficients c_j , $j \in N$ are given. B is the set of indices of binary variables (those that can take on only values zero or one), I is the set of indices of integer variables (those that can take on any integer value), and C is the set of indices of continuous variables. As indicated above, each of the first set of constraints Eq. (2.9) can be either an inequality constraint ("$\leq$" or "$\geq$") or an equality constraint ("="). The data l_j and u_j are the lower and upper bound values, respectively, for variable x_j , $j \in N$.

Feasible sets may be *bounded* or *unbounded.* For example, the feasible set defined by the constraint set ($x \geq 0$, $y \geq 0$) is *unbounded* because in some directions there is no limit on how far one can go and still be in the feasible region. In contrast, the feasible set formed by the constraint set ($x \geq 0, y \geq 0, x + 2y \leq 4$) is *bounded* because the extent of movement in any direction is limited by the constraints.

This general class of problems has many important special cases. When $B = I = \phi$;, we have what is known as a *linear optimization problem* (LP) . If $C = I = \phi$;, then the problem is referred to as a (pure) *binary integer linear optimization problem* (BILP). Finally, if $C = \phi$;, the problem is called a (pure) *integer linear optimization problem* (ILP). Otherwise, the problem is simply an MILP. Throughout this discussion, we refer to the set of points satisfying Eqs. (2.8)-(2.13) as S, and the set of points satisfying all but the integrality restrictions Eqs. (2.11)-(2.12) as P. The problem of optimizing over P with the same objective function as the original MILP is called the LP *relaxation* and arises frequently in algorithms for solving MILPs.

Solution of an MILP involves finding one or more best (optimal) solutions from the set S. Such problems occur in almost all fields of management (e.g., finance, marketing, production, scheduling, inventory control, facility location and layout, supply chain management), as well as in many engineering disciplines (e.g., optimal design of transportation networks, integrated circuit design, design and analysis of data networks, production and distribution of electrical power, collection and management of solid waste, determination of minimum energy states for alloy construction, planning for energy resource problems, scheduling of lines in flexible manufacturing facilities, and design of experiments in crystallography).

2.5.1 Solution Method

Solving integer optimization problems (finding an optimal solution), can be a difficult task. The difficulty arises from the fact that unlike (continuous) linear optimization problems, for which the feasible region is convex, the feasible regions of integer optimization problems consists of either a discrete set of points or, in the case of general MILP, a set of disjoint polyhedra. In solving a linear optimization problem, one can exploit the fact that, due to the convexity of the feasible region, any locally optimal solution is a global optimum. In finding global optima for integer optimization problems, on the other hand, one is required to prove that a particular solution dominates all others by arguments other than the calculus-based approaches of convex optimization. The situation is further complicated by the fact that the description of the feasible region is "implicit" In other words, the formulation Eqs. (2.8)-(2.13) does not provide a computationally useful geometric description of the set S. The general outline of a general solution method is as follows.

1. Identify a (tractable) convex relaxation of the problem and solve it to either
   (a) obtain a valid upper bound on the optimal solution value; or
   (b) prove that the relaxation is infeasible or unbounded (and thus, the original MILP is also infeasible or unbounded).
2. If solving the relaxation produces a solution $\hat{x} \in R^N$ that is feasible to the MILP, then this solution must also be optimal to the MILP.
3. Otherwise, either
   (a) identify a logical disjunction satisfied by all members of S, but not by $\hat{x}$ and add it to the description of P (more on how this is done below); or
   (b) identify an implied linear constraint (called a valid inequality or a cutting plane) satisfied by all members of S, but not by $\hat{x}$ and add it to the description of P.

In Step 1, the LP relaxation obtained by dropping the integrality conditions on the variables and optimizing over P is commonly used. Other possible relaxations include Lagrangian relaxations [37, 42], semi-definite programming relaxations [73], and combinatorial relaxations, e.g., the one-tree relaxation for the traveling salesman problem [47]. Some of the successful MILP solving methods are discussed below.

2.5.1.1 Branch-and-Bound Algorithms

The branch-and-bound method was first proposed in [56] and consists of generating disjunctions satisfied by points in S and using them to partition the feasible region into smaller subsets. Some variant of the technique is used by practically all state-of-the-art solvers. An LP-based branch-and-bound method consists of solving the LP relaxation as in Step 1 above to either obtain a solution and an associated upper bound or to prove *infeasibility* or *unboundedness.*

2.5.1.2 Cutting Plane Algorithms

Cutting plane algorithms was first derived in [46]. A general cutting plane approach relaxes the integrality restrictions on the variables and solves the resulting LP relaxation over the set P. If the LP is unbounded or infeasible, so is the MILP. If the solution to the LP is integer, i.e., satisfies constraints Eqs. (2.11)-(2.12), then one has solved the MILP. If not, then one solves a separation problem whose objective is to find a valid inequality that "cuts off" the fractional solution to the LP relaxation while assuring that all feasible integer points satisfy the inequality, that is, an inequality that "separates" the fractional point from the polyhedron $conv(S)$. Such an inequality is called a "cut" for short. The algorithm continues until termination in one of two ways: either an integer solution is found (the problem has been solved successfully) or the LP relaxation is infeasible and therefore the integer problem is infeasible.

2.5.1.3 Branch and Cut Algorithms

Branch and cut [67] (sometimes written as branch-and-cut) is a method of combinatorial optimization for solving integer linear programs (ILPs), that is, linear programming (LP) problems where some or all the unknowns are restricted to integer values. The method solves the linear program without the integer constraint using the regular simplex algorithm. When an optimal solution is obtained, and this solution has a non-integer value for a variable that is supposed to be integer, a cutting plane algorithm may be used to find further linear constraints which are satisfied by all feasible integer points but violated by the current fractional solution. These inequalities may be added to the linear program, such that resolving it will yield a different solution which is hopefully "less fractional".

At this point, the branch and bound part of the algorithm is started. The problem is split into multiple (usually two) versions. The new linear programs are then solved using the simplex method and the process repeats. During the branch and bound process, non-integral solutions to LP relaxations serve as upper bounds and integral solutions serve as lower bounds. A node can be pruned if an upper bound is lower than an existing lower bound. Further, when solving the LP relaxations, additional cutting planes may be generated, which may be either global cuts, i.e., valid for all feasible integer solutions, or local cuts, meaning that they are satisfied by all solutions fulfilling the side constraints from the currently considered branch and bound subtree.

2.5.2 Applications

Many discrete combinatorial problems have been formulated as MILP. Now we will discuss some of the applications of MILP.

2.5.2.1 Knapsack problems

Suppose one wants to fill a knapsack that has a weight capacity limit of W with some combination of items from a list of n candidates, each with weight w_i and value v_i, in such a way that the value of the items packed into the knapsack is maximized. This problem has a single linear constraint (that the weight of the items selected not exceed W), a linear objective function (to maximize the sum of the values of the items in the knapsack), and the added restriction that each item either be in the knapsack or not. It is not possible to select a fractional portion of an item. For solution approaches specific to the knapsack problem, see [61].

2.5.2.2 Network and graph problems

Many optimization problems can be represented by a network, formally defined as a set of nodes and a set of arcs (uni-directional connections specified as ordered pairs of nodes) or edges (bi-directional connections specified as unordered pairs of nodes) connecting those nodes, along with auxiliary data such as costs and capacities on the arcs (the nodes and arcs together without the auxiliary data form a graph). Solving such network problems involves determining an optimal strategy for routing certain "commodities" through the network. This class of problems is thus known as network flow problems. Many practical problems arising from physical networks, such as city streets, highways, rail systems, communication networks, and integrated circuits, can be modeled as network flow problems. In addition, there are many problems that can be modeled as network flow problems even when there is no underlying physical network. For example, in the assignment problem, one wishes to assign people to jobs in a way that minimizes the cost of the assignment. This can be modeled as a network flow problem by creating a network in which one set of nodes represents the people to be assigned, and another set of nodes represents the possible jobs, with an arc connecting a person to a job if that person is capable of performing that job. A general survey of applications and solution procedures for network flow problems is given in [3].

Space-time networks are often used in scheduling applications. Here, one wishes to meet specific demands at different points in time. To model this problem, different nodes represent the same entity at different points in time. An example of the many scheduling problems that can be represented as a space-time network is the airline fleet assignment problem, which requires that one assign specific planes to pre-scheduled flights at minimum cost ([2]). Each flight must have one and only one plane assigned to it, and a plane can be assigned to a flight only if it is large enough to service that flight and only if it is on the ground at the appropriate Location, Routing, and Scheduling Problems Many network-based combinatorial problems involve finding a route through a given graph satisfying specific requirements. In the Chinese postman problem, one wishes to find a shortest walk (a connected sequence of arcs) through a network such that the walk starts and ends at the same node and traverses every arc at least once ([34]). This models the problem faced by a postal delivery worker attempting to minimize the number traversals of each road segment on a given postal route. If one instead requires that each node be visited exactly once, the problem becomes the notoriously difficult traveling salesman problem ([5]). The traveling salesman problem has numerous applications within the routing and scheduling realm, as well as in other areas, such as the routing of sonet rings ([74]), and the manufacturing of large-scale circuits ([10]). The well-known vehicle routing problem is a generalization in which multiple vehicles must each follow optimal routes subject to capacity constraints in order to jointly service a set of customers ([44]).

A typical scheduling problem involves determining the optimal sequence in which to execute a set of jobs subject to certain constraints, such as a limited set of machines on which the jobs must be executed or a set of precedence constraints restricting the job order (see ([4])). The literature on scheduling problems is extremely rich and many variants of the basic problem have been suggested ([70]).

Location problems involve choosing the optimal set of locations from a set of candidates, perhaps represented as the nodes of a graph, subject to certain requirements, such as the satisfaction of given customer demands or the provision of emergency services to dispersed populations ([33]). Location, routing, and scheduling problems all arise in the design of logistics systems, i.e., systems linking production facilities to end-user demand points through the use of warehouses, transportation facilities, and retail outlets. Thus, it is easy to envision combinations of these classes of problems into even more complex combinatorial problems and much work has been in this direction.

2.5.2.3 Packing, Partitioning, and Covering Problems

Many practical optimization problems involve choosing a set of activities that must either "cover" certain requirements or must be "packed" together so as not exceed certain limits on the number of activities selected. The airline crew scheduling problem, for example, is a covering problem in which one must choose a set of pairings (a set of flight legs that can be flown consecutively by a single crew) that cover all required routes ([50, 86]). Surveys on set partitioning, covering and packing, are given in [9, 17, 52].

2.6 Definition of the main problems

In this section we will define the two problems we are solving in this thesis namely Minimum String Cover (MSC) and Minimum Common String Partition (MCSP) problem.

2.6.1 MSC problem

The input of MSC problem consists of a finite set S (also called Target set) of strings over a symbolic alphabet, and the task is to find a set of substrings generating S and having a smaller cardinality than the alphabet.

Definition 2.13. *MSC*: Given a weight or cost function, $w : C(S) \to \mathbb{N}$, a Minimum String Cover (MSC) is a cover of S with minimum possible total weight. The total weight of the cover C is $W = \sum_{c \in C} w(c)$. Let m be the maximum length of any string of S and $|s|$ be the length of any string s. If weight or cost, $w(c)$, is a unitary function, i.e., $w(c) = 1$ for every $c \in C(S)$, then it minimizes the cardinality of the cover.

Example 2.11. *For example, {AB,C} is the only minimum string cover of {ABC,CAB,CC} while {A,B,C}, {AB,C,ACB}, or {ABC,CAB,ACB} are each minimum string covers for the set of strings {ABC,CAB,ACB}. For non-unit weight functions, the optimal solutions may be different. Given the weight function* $w(t) = |t|$, *{A,B,C} and {AB,C} are the minimum string covers for the first set of strings while {A,B,C} is the minimum string cover for the second set of strings.*

2.6.2 MCSP problem

In the MCSP problem, we are given two *related* strings (X, Y). Clearly, two strings have a common partition if and only if they are related. So, the length of the two strings are also the same (say, n). Our goal is to partition each string into c segments called *blocks*, so that the *blocks* in the partition of X and that of Y constitute the same multiset of substrings. Cardinality of the partition set, i.e., c is to be minimized. As it is defined earlier, a *partition* of a string X is a sequence $P = (B_1, B_2, \ldots, B_m)$ of strings whose concatenation is equal to X, that is $B_1 B_2 \ldots B_m = X$.

Definition 2.14. *MCSP* Given a partition P of a string X and a partition Q of a string Y, we say that the pair $\pi =< P, Q >$ is a common partition of X and Y if Q is a permutation of P. The minimum common string partition problem is to find a common partition of X, Y with the minimum number of blocks.

Example 2.12. *For example, if* $(X,Y) = \{$*"ababcab", "abcabab"*$\}$, *then one of the minimum common partition sets is* $\pi =\{$*"ab", "abc", "ab"*$\}$ *and the minimum common partition size is 3.*

By k-MCSP we denote the version of MCSP where each letter occurs at most k times in each input string.

Example 2.13. *If,* $(X,Y) = \{$*"ababcab", "abcabab"*$\}$, *then the problem is at least 3-MCSP as each letter occurs in the input strings at most 3 times.*

Chapter 3

Related Works

In this chapter we summarizes the previous works on MSC and MCSP problems. The first section is dedicated for describing the available algorithms for MSC problem. The second section detailed the earlier algorithms of the MCSP problem.

3.1 Previous works on MSC Problem

MSCP is perhaps the first problem in Stringology that incorporates the idea of covering. It is surprising that the topic did not come to the surface until 1990 when Neraud [65] studied the problem of determining whether a given set of words is *elementary*, A set of strings is said to be elementary if it does not have a cover of size strictly less than its own size.

MSCP was formally defined and studied in [48], where it was proved to be NP-hard. It was further shown that in general, the problem is hard to be approximated within a factor of $c \cdot \ln n$ for some $c > 0$, and within $\lfloor m/2 \rfloor - 1 - \epsilon$ where $\epsilon > 0$. It was also proved that the problem remains APX-hard even when m is a constant, and the given weight function is either unitary or a length-weighted function. We will now briefly describe some algorithms to solve MSC problem that is found in the literature.

3.1.1 Linear programming formulation

In [48] authors provided an initial linear programming formulation of MSC problem. Given a string s, an l-factorization of s is an ordered multiset of substrings $f = (t_1, \ldots, t_p)$ such that $s = t_1, \ldots, t_p$ and $p \leq l$. Denote by $F_l(s)$ the set of possible l-factorizations of s, and let $F_l(S)$ denote the set of all factorizations of strings in S. Now, for every substring $c \in C(S)$, we designate a variable x_t which associated with t, and for every factorization $f \in F_l(S)$, we designate a variable y_f which is associated with f. In these terms, MSC can be formulated using the following integer linear program:

$$min \sum_{t \in C(S)} w(t) x_t \tag{3.1}$$

s.t.,

$$\sum_{f \in F_l(s)} y_f \geq 1 \;\; \forall s \in S \tag{3.2}$$

$$\sum_{t \in f \in F_l(s)} y_f \leq x_t \;\; \forall s \in S, \forall\, t \text{ suubstring of } S \tag{3.3}$$

$$x_t, y_f \in \{0,1\} \;\; \forall t \in C(S),\; \forall f \in F_l(S) \tag{3.4}$$

3.1.2 Local-Ratio Algorithms

The author in [48] gave a local ratio algorithm for solving MSC problem. The algorithm is polynomial with the approximation ratio $\binom{n+1}{2} - 1$. The algorithm is shown in Algorithm 1

The general outline of algorithm LR is as follows: First, the algorithm adds all substrings $c \in C(S)$ with zero weight to an initial partial-solution C, since these do not have effect on the total weight of the optimal solution. Then, if C is not already a cover of S, LR selects a string $s \in S$ not covered by C, and examines all substrings C_s of s not already in C. It then subtracts $\epsilon = min\{w(t) : t \in C_s\}$ from the weight of all substrings in C_s, and recurses on the new weight function. The last line of the algorithm ensures that at least one substring of s will not be included in C.

Algorithm 1 LR(S, w, l)

```
$C \leftarrow \{t \in C(S) : w(t) = 0\}$
**if** $l$-cover of $S$ **then return** C
**end if**
Let $s \in S$ be a string not $l$-covered by $C$ of maximum length.
$C_s \leftarrow \{t \in C(S)\ /\ C : tis\ a\ substring\ of\ s\}$.
Set $\epsilon = min\{w(t) : t \in C_s\}$
**if** $c \in C_s$ **then**
    $w_1 \leftarrow \epsilon$
**else**
    $w_1 \leftarrow 0$
**end if**
$w_2 = w - w_1$
$C \leftarrow LR(S, w_2, l)$
**if** $C/\ s$ is an $l$-cover of $S$ **then**
    $C \leftarrow C\ /\ s$
**end ifreturn** $C$
```

3.1.3 A polynomial-size ILP formulation

Recently, an alternative flow-based ILP formulation of polynomial size is proposed in [18]. The ILP formulation assumes *substring graph*, $G_s(V_s, E_s)$ is constructed for every string $s \in S$. It uses a binary decision variable $z_{s,i,j}$ for every edge, i.e. interval $(s, i, j) \in I(S)$, and models a path from the source $(s, 0)$ to the sink $(s, |s|)$ as a unit flow. Let $\delta^-(v)$ and $\delta^+(v)$ for the sets of incoming and outgoing edges of $v \in V_s$, respectively. Let $V_s^{\pm} = V_s/\ \{(s,0),(s,|s|)\}$. Using these, the ILP formulation is as follows,

$$min \sum_{t \in C(S)} w(t)x_t \tag{3.5}$$

subject to,

$$z_{s,i,j} \leq x_{s[i...j]}\ \ \forall(s,i,j) \in I(S) \tag{3.6}$$

$$\sum_{(s,i,j) \in \delta^+(s,0)} z_{s,i,j} = 1\ \ \forall s \in S \tag{3.7}$$

$$\sum_{(s,i,j)\in\delta^-(v)} z_{s,i,j} = \sum_{(s,i,j)\in\delta^+(v)} z_{s,i,j} \quad \forall s \in S, \forall v \in V_s^{\pm} \tag{3.8}$$

$$x_t, z_{s,i,j} \in 0,1 \quad \forall t \in C(S), \forall (s,i,j) \in I(S) \tag{3.9}$$

3.1.4 Lagrange based relaxation

The ILP formulation derived exhibits a structure that is favorable for a Lagrangian relaxation approach. The general idea of Lagrangian relaxation is to relax the "complicating" constraints and penalize their violation in the objective function, such that an "easy-to-solve" subproblem remains. In the ILP formulation, solutions satisfying Eqs. (3.7)-(3.8) encode a unit flow and thus a path from node $(s, 0)$ to node $(s, |s|)$ in the *substring graph* of every string $s \in S$. The link between these paths and the chosen substrings is established through Eq. (3.6). Therefore, by relaxing these "linking constraints" and by penalizing their violation with non-negative multipliers λ in the objective function, an optimal solution to the resulting problem is obtained in [18] by computing shortest paths in the substring graphs independently for each string. The new objective function relaxing constraint is now,

$$min \sum_{t\in C(S)} w(t)x_t + \sum_{(s,i,j)\in I(S)} \lambda_{s,i,j}(z_{s,i,j} - x_{s[i\ldots j]}) \tag{3.10}$$

3.2 Previous works on MCSP

MCSP is essentially the breakpoint distance problem [88] between two permutations which is to count the number of ordered pairs of symbols that are adjacent in the first string but not in the other; this problem is obviously solvable in polynomial time [45]. The 2-MCSP is proved to be NP-hard and moreover APX-hard in [45]. The authors in [45] also presented several approximation algorithms. Chen et al. [19] studied the problem, Signed Reversal Distance with Duplicates (SRDD), which is a generalization of MCSP. They gave a 1.5-approximation algorithm for 2-MCSP. In [22], the author analyzed the fixed-parameter tractability of MCSP considering different parametrs. In [54], the authors investigated k-MCSP along with two other variants: $MCSP^c$, where the alphabet size is at most c; and x-balanced MCSP, which requires that the length of the blocks must be witnin the range $(n/d-x, n/d+x)$, where d is the number of blocks in the optimal common partition and x is a constant integer. They showed that $MCSP^c$ is NP-hard when $c \geq 2$. As for k-MCSP, they presented an FPT algorithm which runs in $O^*((d!)^{2k})$ time.

3.2.1 Natural Greedy Approach

Chrobak et al. [20] analyzed a natural greedy heuristic for MCSP: iteratively, at each step, it extracts a longest common substring from the input strings. They showed that for 2-MCSP, the approximation ratio (for the greedy heuristic) is exactly 3. They also proved that for 4-MCSP the ratio would be at least $\Omega(\log n)$ and for the general MCSP, between $\Omega(n^{0.43})$ and $O(n^{0.67})$. The algorithm is listed in Algorithm 2

Algorithm 2 Greedy for MCSP

```
Let $A$ and $B$ be two related input strings
while there are symbols in $A$ or $B$ outside marked blocks do
    $S \leftarrow$ longest common substring of $A$, $B$ that does not overlap previously marked blocks
    mark one occurrence of $S$ in each of $A$ and $B$ as blocks
end while
$(P, Q) \leftarrow$sequence of consecutive marked blocks in $A$ and $B$, respectively
```

Example 3.1. *For example, if A = "cdabcdabceab", B = "abceabcdabcd", then Greedy first marks substring "abcdabc", then "ab", and then three single-letter substrings "c", "d", "e", so the resulting partition is ("c", "d", "abcdabc", "e", "ab"), ("ab", "c", "e", "abcdabc", "d") while the optimal partition is ("cdabcd", "abceab"), ("abceab", "cdabcd"). In this example the problem is 4-MCSP. The approximation ratio for the instance is, $5/2 = 2.5$. The approximation ratio for 4-MCSP with $n = 12$ is at least, $log(12) \approx 1.08$.*

Chapter 4

ACO Algorithm for MSC problem

In this chapter we will develop a Hybrid Ant System (HAS) for solving MSC problem. The details of the algorithm with the pseudocode is described.

4.1 Hybrid Ant System for MSC problem

We have implemented a Hybrid Ant System (HAS) which combines the idea of ACS and MMAS. Here, the construction of a solution is formed according to ACS and the pheromone update is done according to MMAS. The following sections describe the steps of our approach to solve MSCP.

4.1.1 Formulation of *Forward and Backward Substring Graph*

For every string $s \in S$, we define two types of *Substring Graphs*, namely, the *Forward Substring Graph* $G_{Fs} = (V_s, E_{Fs})$ and the *Backward Substring Graph*, $G_{Bs} = (V_s, E_{Bs})$. Here the vertices are the positions of a position of a string. For the *Forward Substring Graph*, two vertices connect if the first vertex is less than the second vertex numerically and vice versa for the *Backward Substring Graph*. The formal definitions of V_s, E_{Fs} and E_{Bs} are given below. We assume, $n = |s|$.

$$V_s = \{(s,0),(s,1),...,(s,n)\} \tag{4.1}$$

$$E_{Fs} = \{((s,p),(s,q)) : p < q, (s,p) \in V_s, (s,q) \in V_s\} \tag{4.2}$$

$$E_{Bs} = \{((s,p),(s,q)) : p > q, (s,p) \in V_s, (s,q) \in V_s\} \tag{4.3}$$

Here an edge $((s,p),(s,q)), p < q$ $(p > q)$ represents the substring interval $(s,p,q-1)$ $((s,q-1,p))$ denoting a substring of length $q-p$ $(p-q)$. Therefore, a *factorization* of s can be represented by a path in G_{Fs} (G_{Bs}) starting from $(s,0)$ $((s,n))$ and ending at (s,n) $((s,0))$. Notably, the idea of *Forward Substring Graph* has been borrowed from [18].

The intuition behind using two types of graphs during the construction of a solution is to favor exploration

4.1.2 Initialization and Configuration

The problem specific heuristic information is contained in η. It is a linear combination of 3 different heuristics for this problem, namely, η_1, η_2 and η_3. Let y be the substring defined by the interval (s_i,p,q). The following equations describe these heuristics for (s_i,p,q):

$$\eta_1(s_i,p,q) = \frac{\sum_{s\in S} \text{frequency of y in s}}{\text{Total number of Distinct Substrings in S}} \tag{4.4}$$

$$G(y,s) = \begin{cases} 1, & \text{if y is a substring of } s \in S \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

$$\eta_2(s_i,p,q) = \frac{\sum_{s\in S} G(y,s)}{|S|} \tag{4.6}$$

$$\eta_3(s_i,p,q) = \frac{\text{Length of This Substring}}{\text{Maximum Allowed Length of Substrings}} \tag{4.7}$$

Here frequency denotes the number of occurrences of a substring in a string. It is expected that, the substrings with higher frequency and greater length would be present in the solution for MSCP. This motivates us to use η_1 and η_3. On the other hand, η_2 is motivated by the assumption that a substring present in higher number of target strings is also expected to be present in the solution of MSCP. Notably, in our experiments a combination of the three (i.e., η_1, η_2 and η_3) has been found to be better than the individual heuristics. We have used the following linear combinations of these heuristics:

$$\eta(s_i,p,q) = (a\cdot\eta_1(s_i,p,q)+b\cdot\eta_2(s_i,p,q)+c\cdot\eta_3(s_i,p,q))/cost(s_i,p,q)/cost(s_i,p,q)$$

We use the following notation. *Local best solution* (L_{LB}) is the best solution found in each iteration. *Global best solution* (L_{GB}) is the best solution found so far among all iterations. The pheromone of the component is bounded between τ_{max} and τ_{min}. Like [84], we use the following values for τ_{max} and τ_{min}: $\tau_{max} = \frac{1}{\varepsilon\cdot cost(L_{GB})}$, and $\tau_{min} = \frac{\tau_{max}(1-\sqrt[n]{p_{best}})}{(avg-1)\sqrt[n]{p_{best}}}$. Here, avg is the average number of choices an ant has in the construction phase. Initially, the pheromone values of all components (substring) are initialized to $initPheromone$ which is a large value to favor the exploration at the first iteration [84].

4.1.3 Construction of a Solution

Let, $nAnts$ denotes the total number of ants in the colony. An ant starts from the ith ($i \in [0,|S|-1]$ is chosen randomly) target string (s_i) to complete a path starting from $(s_i,0)$ $((s_i,|s_i|))$ and ending at $(s_i,|s_i|)$ $((s_i,0))$ in G_{Fs} (G_{Bs}). Let $d(V_k)$ denotes the set of vertices directly reachable from a vertex V_k in the *substring graphs*. So, at any vertex V_k an ant has $|d(V_k)|$ choices. From V_k, the probability of selecting the vertex $V_j \in d(V_k)$ is,

$$p(V_j) = \frac{\tau(V_j)^\alpha\cdot\eta(V_j)^\beta}{\sum_l \tau(V_l)^\alpha\cdot\eta(V_l)} \tag{4.8}$$

Let V_{knext} be the next vertex to be chosen, i.e., interval $(s_i, k, knext)$ is selected as a component of the factorization of s_i. Then,

$$V_{knext} = \begin{cases} argmax_{V_j\in d(V_k)}\ p(V_j), & \text{if } q\le q_0 \\ \text{drawprob} & \text{otherwise}\end{cases} \tag{4.9}$$

Here, q is a random number taken from a uniform distribution, q_0 is the threshold value and *drawProb* represents a vertex selected randomly from the probability distribution function in Eq. (4.8). Thus a factorization of target string s_i is completed. An ant chooses whether to factor according to *forward substring graph* or *backward substring graph* with equal probability. This process continues until an ant finishes the factorization of all target strings. In each iteration, $nAnts$ solutions are constructed by the ants.

The intuition behind using two types of graphs during the construction of a solution is to favor exploration. An ant starting from the initial position may not fully explored the solution space. Again starting from each vertex randomly can be effective but very inefficient and impractical due to high time complexity. So, we have chosen only two types of starting either from the first vertex or the last vertex. The starting positions are chosen randomly. If the first vertex is chosen as starting position, then the factorization of the string is done on *forward substring graph* otherwise the factorization is done upon *backward substring graph.*

4.1.4 Pheromone Update

We have defined the fitness $F(L)$ of a solution L as the reciprocal of the sum of the costs of the interval in L. The pheromone of each interval of each target string is computed according to Eq. (2.7) after each iteration. Now, like MMAS, the pheromones are bounded within the range τ_{MIN} and τ_{MAX}. We have updated the pheromone values according to L_{LB} or L_{GB}. The algorithm is listed in 3

4.1.5 Pseudocode

The pseudocode of our approach for solving MSCP is given in Algorithm 4.

Algorithm 3 Update pheromone for MSC problem

```
if $run \leq 50$ then
    Update by $L_{GB}$
else if $run \leq 100$ then
    if $run\%5 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 200$ then
    if $run\%4 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 400$ then
    if $run\%3 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 800$ then
    if $run\%2 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else
    Update by $L_{LB}$
end if
```

Algorithm 4 HAS for MSCP

```
Calculate heuristic information()
for $run = 1 \to MAXRUN$ do
    Initialize pheromone
    Initialize global best
    repeat
        Initialize local best
        for $ant = 1 \to MAXPOPULATION$ do
            Construction for $ant_i$
            update local best
        end for
        update global best
        update pheromone either by local best or global best
    until time reaches $maxAllowedTime$ or No update found for $maxAllowedIteration$
end for
```

Chapter 5

Algorithms for solving MCSP

In this Chapter we will present 2 (two) algorithms for solving the MCSP. The first part of this chapter will focus in designing a Mixed Integer Programming formulation of MCSP. The second one will present a MAX-MIN Ant System.

5.1 MILP formulation of MCSP

Given two *related* strings X and Y each of length n, we create two graphs namely, $G_{cs}(V_1, E_1, id(X))$ and $G_{cs}(V_2, E_2, id(Y))$ of (X, Y), where V_1 and V_2 are the vertex sets and E_1 and E_2 denote two edge block sets from the two graphs respectively. For each $t \in E_1$ and $t \in E_2$, we define two sets of binary variables, namely, X_t and Y_t. We also write $\delta_k(v)^-$ and $\delta_k(v)^+$ for the sets of incoming and outgoing edge blocks from E_k where $v \in V_k$. With the above settings, we develop the MIP formulation for the MCSP as follows:

$$min \sum_{t\in[0,n-1]} (X_t + Y_t)/2 \tag{5.1}$$

subject to,

$$\sum_{t\in[0,n-1]} X_t = \sum_{t\in[0,n-1]} Y_t \tag{5.2}$$

$$\sum_{t\in\delta_1(0)^+} X_t = 1 \tag{5.3}$$

$$\sum_{t\in\delta_1(v)^-} X_t = \sum_{t\in\delta_1(v+1)^+} X_t \;\; \forall_{v\in[0,n-1]} \tag{5.4}$$

$$\sum_{t\in\delta_2(0)^+} Y_t = 1 \tag{5.5}$$

$$\sum_{t\in\delta_1(v)^-} Y_t = \sum_{t\in\delta_1(v+1)^+} Y_t \;\; \forall_{v\in[0,n-1]} \tag{5.6}$$

$$X_t \leq \sum b \in matchList(1,t) Y_t \;\; \forall_{t\in E_1} \tag{5.7}$$

$$Y_t \leq \sum b \in matchList(1,t) X_t \;\; \forall_{t\in E_2} \tag{5.8}$$

5.1.1 Explanation of the Formulation

Objective function: Eq. (5.1) is the objective function that is to be minimized. The function simply calculate the size of the partition.

Equality constraint: Eq. (5.2) states that two partition on the two substring graphs must be equal in size. That is the number of blocks in the factorization of the first string X is equal with the number of blocks in the factorization of the second string Y.

Factorization constraint: Eqs. (5.3)-(5.4) together imply that a unit flow enters at the source (the vertex labeled with 0) and arrives at the sink (the vertex labeled with $n-1$) for string x. So, the string is factorized. For string y the factorization is achieved in a similar fashion by Eqs.(5.5)-(5.6). These constraints ensure that the srtings get factorized by non-overlapping blocks.

One to one match constraint: Now that we have factorized the strings, we need to achieve one to one matching between the selected blocks. We have two sets of blocks after the factorization. We must ensure that there is a one to one matching between the two sets of blocks. By matching we mean that, for each selected blocks (those with $X_t = 1$) of the first edge block set E_1, there must be one and only one corresponding selected block (with $Y_t = 1$) with the same substring in the second edge block set E_2 and vice versa. The constraint is achieved by Eqs. (5.7)-(5.8).

5.2 MAX-MIN Ant System for MCSP

In this section we will develop a MAX-MIN Ant System for solving MCSP problem. At first we will define the *Common Substring Graph*. Upon this graph, the MAX-MIN Ant System will be developed.

5.2.1 Formulation of the *Common Substring Graph*

We define a common substring graph, $G_{cs}(V, E, id(X))$ of a string X with respect to Y as follows. Here V is the vertex set of the graph and E is the edge set. Vertices are the positions of string X, i.e., for each $v \in V$, $v \in [0, |X| - 1]$. Two vertices $v_i \leq v_j$ are connected with an edge, i.e, $(v_i, v_j) \in E$, if the substring induced by the block $[id(X), v_i, v_j]$ matches some substrings of Y. More formally, we have:

$$(v_i, v_j) \in E \Leftrightarrow cstring([id(X), v_i, v_j], B') \text{ is not empty } \exists B' \in Y$$

In other words, each edge in the edge set corresponds to a *block* satisfying the above condition. For convenience, we will denote the edges as *edge blocks* and use the list of edge blocks (instead of edges) to define the edgeset E. Notably, each *edge block* on the edge set of $G_{cs}(V, E, id(X))$ of string (X, Y) may match with more than one blocks of Y. For each *edge block* B a list is maintained containing all the matched blocks of string Y to that *edge block*. This list is called the *matchList*(B).

Example 5.1. *For example, suppose* $(X, Y) = \{$ *"abcdba", "abcdab"*$\}$. *Now consider the corresponding common substring graph. Then, we have* $V = \{0, 1, 2, 3, 4, 5\}$ *and* $E = \{[0,0,0], [0,0,1], [0,1,1], [0,0,2], [0,0,3], [0,1,2], [0,1,3], [0,2,2], [0,2,3],$ $[0,3,3][0,4,4], [0,5,5]\}$. *The matchList of the second* edge block, *i.e., matchList*$([0,0,1]) = \{[1,0,1], [1,4,5]\}$.

To find a common partition of two strings (X, Y) we first construct the common substring graph of (X, Y). Then from a vertex v_i on the graph we take an edge block $[id(X), v_i, v_j]$. Suppose M_i is the *matchList* of this block. We take a block B'_i from M_i. Then we advance to the next vertex that is $(v_j + 1)\ MOD\ |X|$ and choose another corresponding edge block as before. We continue this until we come back to the starting vertex. Let *partitionList* and *mappedList* are two lists, each of length c, containing the traversed edge blocks and the corresponding matched blocks. Now we have the following lemma.

Lemma 1. *partitionList* is a common partition of length c iff,

$$B_i \cap B_j = [] \ \forall B_i, B_j \in mappedList,\ i \neq j \tag{5.9}$$

and

$$B_1 \cup B_2 \cup \cdots \cup B_c = [id(Y), 0, |Y| - 1] \tag{5.10}$$

Proof. By construction, *partitionList* is a *partition* of X. We need to prove that *mappedList* is a partition of Y and with the one to one correspondence between *partitionList* and *mappedList* it is obvious that *partitionList* would be the common partition of (X, Y). Eq. (5.9) asserts the non overlapping property of *mappedList* and Eq. (5.10) assures the cover property. So, *mappedList* will be a partition of Y if Eqs. (5.9)-(5.10) are satisfied.

On the other hand let *partitionList* along with *mappedList* is a common partition of (X, Y). According to construction, *partitionList* satisfies the two properties of a partition. Let, *mappedList* is a partition of Y. We assume *mappedList* does not follow the Eqs. (5.9)-(5.10). So, there might be overlapping between the blocks or the blocks do not cover the string Y, a contradiction. This completes the proof.

□

5.2.2 Heuristics

Heuristics (η) contain the problem specific information. We propose two different (types of) heuristics for MCSP. Firstly, we propose a static heuristic that does not change during the iterations of algorithm. The other heuristic we propose is dynamic in the sense that it changes between the iterations.

5.2.2.1 The Static Heuristic for MCSP

We employ an intuitive idea. It is obvious that the larger is the size of the blocks the smaller is the partition set. To capture this phenomenon, we assign on each edge of the common substring graph a numerical value that is proportional to the length of the substring corresponding to the edge block. Formally, the static heuristic (η_s) of an edge block $[id, i, j]$ is defined as follows:

$$\eta_s([id, i, j]) \propto length([id, i, j]) \tag{5.11}$$

5.2.2.2 The Dynamic Heuristic for MCSP

We observe that the static heuristic can sometimes lead us to very bad solutions.

Example 5.2. *For example if* $(X, Y) = \{$*"bceabcd", "abcdbec"*$\}$ *then according to the static heuristic much higher value will be assigned to* edge block $[0, 0, 1]$ *than to* $[0, 0, 0]$. *But if we take* $[0, 0, 1]$, *we must match it to the block* $[1, 1, 2]$ *and we further miss the opportunity to take* $[0, 3, 6]$ *later. The resultant partition will be* $\{$*"bc", "e", "a", "b", "c", "d"*$\}$ *but if we would take* $[0, 0, 0]$ *at the first step, then one of the resultant partitions would be* $\{$*"b", "c", "e", "abcd"*$\}$.

To overcome this shortcoming of the static heuristic we define a dynamic heuristic as follows. The dynamic heuristic (η_d) of an edge block ($B = [id, i, j]$) is inversely proportional to the difference between the length of the block and the minimum span of its corresponding blocks in its *matchList*. More formally, $\eta_d(B)$ is defined as follows:

$$\eta_d(B) \propto \frac{1}{|length(B) - minSpan(B)| + 1}, \tag{5.12}$$

where

$$minSpan(B) = \min\{span(B') \mid B' \in matchList(B)\} \tag{5.13}$$

In the example 5.2, $minSpan([0, 0, 0])$ is 1 as follows: $matchList([0, 0, 0]) = \{[1, 1, 1], [1, 4, 4]\}$. $span([1, 1, 1]) = 4$ and $span([1, 4, 4] = 1)$. On the other hand, $minSpan([0, 0, 1])$ is 4. So, according to the dynamic heuristic much higher numeral will be assigned to block $[0, 0, 0]$ rather than to block $[0, 0, 1]$.

We define the total heuristic (η) to the linear combination of the static heuristic (η_s) and the dynamic heuristic (η_d). Formally, the total heuristic of an edge block B is,

$$\eta(B) = a \cdot \eta_s(B) + b \cdot \eta_d(B) \tag{5.14}$$

where a, b are any real valued constants.

5.2.3 Initialization and Configuration

Given two strings (X, Y), we first construct the common substring graph $G_{cs} = (V, E, id(X))$. We use the following notations. *Local best solution* (L_{LB}) is the best solution found in each iteration. *Global best solution* (L_{GB}) is the best solution found so far among all iterations. The pheromone of the edge block is bounded between τ_{max} and τ_{min}. Like [84], we use the following values for τ_{max} and τ_{min}: $\tau_{max} = \frac{1}{\varepsilon \cdot cost(L_{GB})}$, and $\tau_{min} = \frac{\tau_{max}(1 - \sqrt[n]{p_{best}})}{(avg-1)\sqrt[n]{p_{best}}}$. Here, avg is the average number of choices an ant has in the construction phase; n is the length of the string; p_{best} is the probability of finding the best solution when the system converges and ε is the evaporation rate. Initially, the pheromone values of all edge blocks (substring) are initialized to $initPheromone$ which is a large value to favor the exploration at the earlier iterations [84].

5.2.4 Construction of a Solution

Let, $nAnts$ denotes the total number of ants in the colony. Each ant is deployed randomly to a vertex v_s of G_{cs}. A solution for an ant starting at a vertex v_s is constructed by the following steps:

step 1: Let $v_i = v_s$. Choose an *available* edge block starting from v_i by the discrete probability distribution defined below. An edge block is available if its *MatchList* is not empty and inclusion of it to the *partitionList* and *mappedList* obeys Eq. (5.15). The probability for choosing edge block $[0, v_i, v_j]$ is:

$$p([0,v_i,v_j]) = \frac{\tau([0,v_i,v_j])^\alpha \cdot \eta([0,v_i,v_j])^\beta}{\sum_\ell \tau([0,v_i,v_\ell])^\alpha \cdot \eta([0,v_i,v_\ell])^\beta}, \forall \ell \text{ such that } [0,v_i,v_l] \textit{ is } \text{an available block.} \tag{5.15}$$

step 2: Suppose, $[0,v_i,v_k]$ is chosen according to Eq. (5.15) above. We choose a match block B_m from the *matchList* of $[0,v_i,v_k]$ and delete B_m from the *matchList*. We also delete every block from every *matchList* of every edge block that overlaps with B_m. Formally we delete a block B if

$$B_m \cap B \neq [] \quad \forall B_i \in E, B \in matchList(B_i).$$

We add $[0,v_i,v_k]$ to the *partitionList* and B_m to the *mappedList*.

step 3: If $(v_k+1)\ MOD\ |X| = v_s$ and the *mappedList* obeys Eq. (5.10), then we have found a common partition of X and Y. The size of the partition is the length of the *partitionList*. Otherwise, we jump to the *step 1*.

5.2.5 Intelligent Positioning

For every edge block of G_{cs} in X, we have a *matchList* that contains the matched block of string Y. In the construction step (step 1), when an edge block is chosen by the probability distribution, we take a block from the *matchList* of the chosen edge block. We can choose the matched block randomly. But we observe that random choosing may lead to a very bad partition.

Example 5.3. *For example, if $(X,Y) = \{$ "ababc", "abcab"$\}$ then the* $matchList([0,0,1]) = \{[1,0,1],[1,3,4]\}$. *If we choose the first match block then eventually we will get the partition as $\{$ "ab", "ab", "c"$\}$ but a smaller partition exists and that is $\{$ "ab", "abc"$\}$.*

To overcome this problem, we have imposed a rule for choosing the matched block. We will select a block from the *matchList* having the lowest possible span. Formally, for the edge block, B_i, a block $B' \in matchList(B_i)$ will be selected such that $span(B')$ is the minimum.

In the example 5.3, $span([1,0,1]) = 3$ where as $span([1,3,4]) = 2$. So it is better to select the second block so that we do not miss the opportunity to match a larger block.

5.2.6 Pheromone Update

When each of the ants in the colony has constructed a solution (i.e., a common partition), an iteration completes. We set the local best solution as the best partition that is the minimum length partition in an iteration. The global best solution for n iterations is defined as the minimum length common partition over all the n iterations.

We define the fitness $F(L)$ of a solution L as the reciprocal of the length of L. The pheromone of each interval of each target string is computed according to Eq. 2.7 after each iteration. The pheromone values are bounded within the range τ_{MIN} and τ_{MAX}. We update the pheromone values according to L_{LB} or L_{GB}. Initially for the first 50 iterations we update pheromone by only L_{LB} to favor the search exploration. After that we develop a scheduling where the frequency of updating with L_{LB} decreases and L_{GB} increases to facilitate exploitation. The pseudocode for pheromone update is given in Algorithm 5

5.2.7 The Pseudocode

The pseudocode of our approach for solving MCSP is given in Algorithm 6.

Algorithm 5 update pheromone for MCSP

```
if $run \leq 50$ then
    Update by $L_{GB}$
else if $run \leq 100$ then
    if $run\%5 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 200$ then
    if $run\%4 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 400$ then
    if $run\%3 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else if $run \leq 800$ then
    if $run\%2 == 0$ then
        Update by $L_{LB}$
    else
        Update by $L_{GB}$
    end if
else
    Update by $L_{LB}$
end if
```

Algorithm 6 MMAS for MCSP

```
Calculate heuristic information()
for $run = 1 \rightarrow MAXRUN$ do
    Initialize pheromone
    Initialize global best
    repeat
        Initialize local best
        for $i = 1 \rightarrow nAnts$ do
            Construction for $ant_i$
            update local best
        end for
        update global best
        update pheromone                    ▷ either by local best or global best
    until time reaches $maxAllowedTime$ or No update found for $maxAllowedIteration$
end for
```

Chapter 6

Experimentation and Results

In this chapter, we will discuss the implementation of our algorithms. The results and analyses are also reported in this chapter. First we will discuss about the simulation evnivornment and the dataset used in the experimentations and then the result will be analyzed.

6.1 Resource Description

The experiments have been conducted using four computers all of which have the same configuration. The compute configuration is listed below,

- Processor: Intet(R) core(TM) i5-2450 CPU @2.50GHz 2.50Ghz
- Main Memory: 4 GB
- L2-Cache: 3 MB
- Operating System: Windows 7 64 bit

The HAS algorithm for MSC problem is implemented in C++ using visual studio 2008 while the MMAS for MCSP problem is implemented using java. The jdk version is "1.7.0_15". The programming environment is "jcreator". The MIP solution for MCSP is implemented in Matlab 2009.

6.2 Dataset

As there are very few practical work on the two problems we are dealing with, we have faced a scarcity of the dataset. For both the problem we have to create our own benchmark to test our algorithm. In this section, we will discuss the dataset we have used.

6.2.1 MSCP

As MSCP has not got enough attention in the literature, there is no standard library of data sets for which the exact solutions are known. Here, we rely on the dataset introduced in [18]. These datasets are described below.

6.2.1.1 Large Random Data

Large Random Datasets has been used in [18]. Here random instances have been generated with alphabet sizes 4, 20 and 50. The number of target strings was 10 and 100. The length of a target string was 50 – 100 and 250 – 300 respectively. The target strings are constructed by randomly concatenating strings from the solution set. The size of solution set was 2 – 10 and 20 – 30 each string having length 3 – 10 and 20 – 30 respectively. A total of 48 different kinds of data for all possible combinations of these parameters were generated. Each kind of data has 50 instances of the same type. The authors in [18] follow a specific naming scheme to describe a particular data-set instance. For example, *instances*_100_*a*4_*s*20-30_*l*20-30_*L*250-300 denotes an instance having 100 target strings, alphabet size (a) 4, solution set size (s) between 20-30, length of solution set (l) between 20-30 and length of target string (L) between 250-300.

6.2.2 MCSP

We have conducted our experiments for MCSP problem on two types of data, namely, randomly generated DNA sequences and real gene sequences.

6.2.2.1 Random DNA sequences

We have generated 30 random DNA sequences each of length at most 600 using [81]. The fraction of bases A, T, G and C is assumed to be 0.25 each. For each DNA sequence we shuffle it to create a new DNA sequence. The shuffling is done using the online toolbox [87]. The original random DNA sequence and its shuffled pair constitute a single input (X, Y) in our experiment. This dataset is divided into 3 classes. The first 10 have lengths less than or equal to 200 bps (base-pairs), the next 10 have lengths within $[201, 400]$ and the rest 10 have lengths within $[401, 600]$ bps.

6.2.2.2 Real Gene Sequences

We have collected the real gene sequence data from the NCBI GenBank[1]. For simulation, we have chosen Bacterial Sequencing (part 14). We have taken the first 15 gene sequences whose lengths are within $[200, 600]$.

6.3 Options and Parameters

During the experiments a lot parameters are to be set for both ACO based algorithms and for the MIP solution. Now we will report the options and parameters used in the experiments.

6.3.1 ACO based Algorithms

There are several parameters which have to be carefully set to tune our ACO based algorithms, i.e., HAS and MMAS. The settings of parameters for which we achieved the results are described in Tables 6.1 and 6.2. These parameters were found based on some preliminary experiments.

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

TABLE 6.1: Parameters for MSC

Parameters	**Value**
α	5.0
β	20.0
Evaporation rate, ε	0.02
$nAnts$	10
q_0	0.25
p_{best}	0.09
$initPheromone$	10.0
avg	$m/2$
Maximum Allowed Time	100 min
Maximum Allowed Length of Substrings	$m/2$ *or* 50
Minimum Allowed Length of Substrings	3 *or* 6
Coeff. of η_1,a	0.01
Coeff. of η_2,b	0.99
Coeff. of η_1,c	1.00

TABLE 6.2: Parameters for MCSP

Parameters	**Value**
α	2.0
β	5.0
Evaporation rate, ε	0.02
$nAnts$	$\|X\|$
p_{best}	0.09
$initPheromone$	10.0
Maximum Allowed Time	120 min
Coeff. of η_s, a	0.5
Coeff. of η_d, b	0.5

6.3.2 MIP

We have used GLPK (GNU Linear Programming Kit) [1] for solving MIP. GLPK is a set of routines written in the ANSI C programming language and organized in the form of a callable library. It is intended for solving linear programming (LP), mixed integer programming (MIP), and other related problems. The raw glpk library is written in C. We have used a third party tool "glpkmex" [43]. It is a Matlab Mex interface for the GLPK library. The options those were used in our experiment is listed in Table 6.3.

A number of parameters are to be set. We have left most of the parameters with default value provided in the package(as it is advised in the document of the package). Some of the important parameters are as follows:

TABLE 6.3: Options for MCSP-MIP

Options	Value
Scaling option	Equilibration scaling
Dual simplex option	Do not use
Pricing option	Steepest edge pricing
Ratio test Technique	Harris's two-pass ratio test
Solution rounding option	Report all primal and dual values "as is"
Simplex iterations limit	No limit
Maximum Allowed Time	120 min
LP solver	Revised Simplex Method
Branching heuristic option	Branch on the most fractional variable.
Backtracking heuristic option	best local bound
Pre-processing technique option	perform preprocessing for root only
usecuts	Gomoy's mixed integer cuts
Binarizeation option	do not use

1. **relax**: 0.07. Relaxation parameter used in the ratio test.
2. **tolbnd**: 10e-7. Relative tolerance used to check if the current basic solution is primal feasible.
3. **toldj**: 10e-7. Absolute tolerance used to check if the current basic solution is dual feasible.
4. **tolpiv**: 10e-9. Relative tolerance used to choose eligible pivotal elements of the simplex table.
5. **tmlim**: 10800 sec. searching time limit, in seconds.
6. **tolint**: 10e-5. Relative tolerance used to check if the current basic solution is integer feasible.
7. **tolobj**: 10e-7. Relative tolerance used to check if the value of the objective function is not better than in the best known integer feasible solution.
8. **mipgap**: 0.0. The relative mip gap tolerance. If the relative mip gap for currently known best integer feasible solution falls below this tolerance, the solver terminates the search. This allows obtaining suboptimal interger feasible solutions if solving the problem to optimality takes too long.

6.4 Results and Analysis

In this section we will analyzed the experimentation results. First we will discuss the results of MCSP and then we will discuss the results and analysis of MCSP implementation.

6.4.1 MSC

There are 2 approaches described in [18]. They are Lagrangian relaxation approach and CPLEX based optimization. Between them, the Lagrangian relaxation was reported to have achieved better results [18]. Therefore, we provide the comparison with only the latter. Please recall that we assume a unitary cost function, i.e., the cost of every substring is considered to be 1.

6.4.1.1 Large Random Data

The data set introduced in [18] contains 48 types of data set each having 50 instances. We chose 10 instances from each type of data set for our experiments. We believe that as the 50 instances are of same characteristics, choosing 10 among them is enough to get an indicative performance. We have run 3 times for every instances we mentioned in the Tables 6.7 and 6.8 and report the best results.

6.4.1.2 Choice of coefficients of the heuristics

We have conducted a preliminary experiment with different choices of a, b and c. The value of a, b and c is taken from the set $\{.01,0.8\}$, $\{0.99,0.1\}$ and $\{1,0.1\}$. The combination with $a = .01$, $b = 0.99$ and $c = 1$ represents the best among the 8 combinations. The preliminary experiment was conducted on the instance "instances_100_a20_s2-10_l20-30_L50-100". The result is presented in Table 6.4

6.4.1.3 Effects of different heuristics

We have defined 3 types of heuristics namely η_1, η_2 and η_3. We have used a linear combination of these 3 heuristics. In this section we will include some preliminary

TABLE 6.4: Average cover size found using different combination of a, b and c of the instance "instances_100_a20_s2-10_l20-30_L50-100"

Instance No	**{0.1,0.99,1}**	**{0.1,0.99,0.1}**	**{0.1,0.1,1}**	**{0.1,0.1,0.1}**	**{0.8,0.99,1}**	**{0.8,0.99,0.1}**	**{0.8,0.1,1}**	**{0.8,0.1,0.1}**
1	4	5	111.5	4	4	5	110.3	4
6	6	6	193.9	8	6	6	193.1	6
11	5	5	147	5	7	5	146.8	5
16	7	7	193.9	7	7	9	197.9	7
21	9	14.5	233.4	14.5	14.28571	9	235	13.85714

experiment with different combination of these 3 heuristics. For experiment we have used the instance "instances_100_a50_s2-10_l20-30_L50-100". Table 6.5 shows the average minimum cover found using different linear combination of the three heuristics. The constant a, b and c are listed in Table 6.1. The linear combination of the three heuristics $(a \cdot \eta_1 + b \cdot \eta_2 + c \cdot \eta_3)$ has given much better results than the other combinations.

6.4.1.4 Observations

Lagrangian relaxation approach aborts after running for 1 hour for 49 instances among the 50 instances of the type with alphabet size 4, solution size $20 - 30$, solution string length $20-30$ and 100 target strings each of length $250-300$. HAS has successfully completed all these cases. We have allowed different maximum lengths of substrings to be used to cover the target strings. For all these cases, we have been able to produce the solutions within at most 15 minutes. This is true even for the cases when HAS needs to work with about 1.5 million substrings when we allowed maximum lengths of substrings to be 100 and the minimum length to be 6. Among the above 50 cases, solutions having size equal to the actual solution size was found in 31 cases and the maximum deviation of cost of our solution from solution size is 5 and that too happened for only 1 case. The average deviation of cost from solution size is only 0.92 for this type.

As can be seen from the results, we have achieved very good results for almost all the cases. Table 6.6 reports the results of 5 instances where HAS has surpassed the Lagrangian relaxation. These results are found for cases with 10 target strings where the solution mentioned in [18] contains $20 - 30$ strings. We have allowed substrings with length $[3, m/2]$ which is a larger search space compared to Lagrangian relaxation approach in [18]. It can be recalled that m denotes the maximum length of a target string.

Tables 6.7 and 6.8 contain the results of 410 instances of 41 types out of the 48 types. We have found solutions which are as good as the solutions reported in [18] for 300 instances and found better solutions for 45 instances. The rest of the instances, which are only about 15% of them, were solved with an average deviation of cost 2 to 3 from the solutions reported in [18]. The boldfaced numbers in Table 6.7 show where HAS has surpassed the solution found by Lagrangian relaxation approach [18].

TABLE 6.5: Average cover size found using different combination of heuristics of the instance "instances_100_a50_s2-10_l20-30_L50-100"

Instance No	**Lagrange**	$a\eta 1 + b\eta_2 + c\eta_3$	η_1	η_2	η_3	$a\eta_1 + b\eta_2$	$b\eta_2 + c\eta_3$	$a\eta_1 + c\eta_3$
3	10	10	153.4	135.8	135.8	148.4	44.9	254.7778
8	10	10	128	125.8	125.8	125.1	44	240
13	3	3	63.9	70.5	70.5	58.4	3	52
18	3	3	76.1	78.1	78.1	76.8	3	112.5
23	6	6	100.9	96.3	96.3	102	6	164.3
28	10	10	120.1	124.1	124.1	117.9	22.8	249.4
33	5	5	75.6	79	79	80.1	5	166
38	8	8	118.3	112	112	108.2	8.6	166
43	8	8	81.6	80.2	80.2	78.8	8	166
48	2	2	48.5	65.25	65.25	62.1	2	166

TABLE 6.6: Analysis of the quality of HAS for 5 instances. Here, No. of Better Res. means the number of times HAS has surpassed Lagrangian relaxation [18], Max. Diff. stands for maximum difference from Lagrangian relaxation among 10 cases and Avg. Diff. means average difference from Lagrangian relaxation for those No. of Better Res. cases.

Instance Description	**No. of Better Res.**	**Max. Diff.**	**Avg. Diff.**
instances_10_a4_s20-30_l3-10_L50-100	10	9	6.0
instances_10_a4_s20-30_l20-30_L50-100	10	13	8.0
instances_10_a20_s20-30_l3-10_L50-100	3	5	3.667
instances_10_a20_s20-30_l20-30_L50-100	10	13	6.4
instances_10_a50_s20-30_l20-30_L50-100	10	11	6.3

6.4.2 MCSP

Throughout the thesis we have developed two approaches for solving MCSP problem. Firstly, we have developed a MILP formulation. From now on the MILP formulation will be referred as MIP. The other is a MAX-MIN Ant System (MMAS). we have compared our approaches with the greedy algorithm of [20] because none of the other algorithms in the literature are for general MCSP: each of the other approximation algorithms put some restrictions on the parameters.

6.4.2.1 Random DNA sequence

Table 6.9, 6.10 and 6.11 present the comparison between our approaches and the greedy approach [20] for the random DNA sequences. For a particular DNA sequence, the experiment was run 15 times and the average result is reported. The first column under any group reports the partition size computed by the greedy approach, the second column is the average partition size found by MMAS, the third and fourth column report the worst and best results among 15 runs, the fifth column represents the difference between the two approaches. A positive (negative) difference indicates that the greedy result is better (worse) than the MMAS result by that amount. The sixth column reports the standard deviation of 15 runs of MMAS, the seventh column is the average time in second by which the reported partition size is achieved. The eighth column is the partition size by MIP approach. The ninth column is the difference between the greedy and MIP and the tenth column is the difference between MIP and MMAS. A negative (positive) difference indicates that the MIP result is better (worse) than the MMAS or Greedy result by that amount.

TABLE 6.7: Comparison Between Lagrange [18] and HAS approach for 10 target strings. "instance no" represents the instance indices we have run, "Lagrange" means results of that instance by Lagrangian relaxation approach [18], "HAS" represents same instance run by our approach. "Diff" represents the difference between Lagrangian relaxation approach and HAS approach. where $(+n)$ means $Solution_{HAS} = Solution_{Lagrange} + n$ and $(-n)$ means $Solution_{HAS} = Solution_{Lagrange} - n$.

instance no	Lagrange	HAS	Diff	instance no	Lagrange	HAS	Diff	instance no	Lagrange	HAS	Diff
instances_10_a4_s2-10_l20-30_L50-100				instances_10_a4_s2-10_l20-30_L250-300				instances_10_a4_s20-30_l3-10_L50-100			
5	5	5	0	1	3	4	1	5	28	**20**	-8
10	10	11	1	6	3	3	0	10	29	**20**	-9
15	4	4	0	11	9	9	0	15	28	**20**	-8
20	4	4	0	16	10	10	0	20	29	**20**	-9
25	2	3	1	21	10	10	0	25	26	**20**	-6
30	5	5	0	26	5	7	2	30	21	**20**	-1
35	9	9	0	31	4	4	0	35	24	**20**	-4
40	6	6	0	36	4	4	0	40	26	**20**	-6
45	3	3	0	41	10	10	0	45	22	**20**	-2
50	8	8	0	46	6	6	0	50	27	**20**	-7
instances_10_a4_s20-30_l20-30_L50-100				instances_10_a4_s20-30_l20-30_L250-300				instances_10_a20_s2-10_l3-10_L50-100			
3	30	**19**	-11	1	25	28	3	2	3	3	0
8	23	**17**	-6	6	25	26	1	7	9	9	0
13	30	**17**	-13	11	22	23	1	12	2	2	0
18	26	**18**	-8	16	20	20	0	17	4	4	0
23	28	**17**	-11	21	22	22	0	22	9	9	0
28	20	**19**	-1	26	29	31	2	27	7	7	0
33	30	**17**	-13	31	24	30	6	32	2	2	0
38	21	**20**	-1	36	30	32	2	37	3	3	0
43	26	**16**	-10	41	30	35	5	42	6	6	0
48	23	**17**	-6	46	20	24	4	47	6	6	0
instances_10_a20_s2-10_l3-10_L250-300				instances_10_a20_s2-10_l20-30_L50-100				instances_10_a20_s2-10_l20-30_L250-300			
1	10	10	0	3	2	2	0	2	7	7	0
6	7	7	0	8	10	10	0	7	6	6	0
11	6	6	0	13	2	3	1	12	6	6	0
16	9	9	0	18	6	6	0	17	5	5	0
21	4	4	0	23	9	9	0	22	2	5	3
26	9	9	0	28	10	10	0	27	9	9	0
31	3	3	0	33	7	**7**	0	32	3	10	7
36	3	3	0	38	4	4	0	37	10	10	0
41	7	**7**	0	43	5	5	0	42	7	**7**	0
46	10	10	0	48	9	9	0	47	7	7	0
instances_10_a20_s20-30_l3-10_L50-100				instances_10_a20_s20-30_l3-10_L250-300				instances_10_a20_s20-30_l20-30_L50-100			
2	24	24	0	1	21	21	0	5	30	**19**	-11
7	26	26	0	6	30	30	0	10	23	**19**	-4
12	30	**28**	-2	11	24	24	0	15	28	**20**	-8
17	27	**23**	-4	16	29	29	0	20	22	**18**	-4
22	25	25	0	21	24	24	0	25	27	**17**	-10
27	23	25	2	26	28	28	0	30	21	**17**	-4
32	29	**24**	-5	31	25	25	0	35	20	**17**	-3
37	26	30	4	36	30	30	0	40	23	**19**	-4
42	21	22	1	41	23	23	0	45	29	**16**	-13
47	23	23	0	46	27	27	0	50	21	**18**	-3
instances_10_a20_s20-30_l20-30_L250-300				instances_10_a50_s2-10_l3-10_L50-100				instances_10_a50_s2-10_l3-10_L250-300			
5	28	28	0	2	5	8	3	1	6	6	0
10	30	30	0	4	9	9	0	2	9	9	0
15	22	22	0	6	4	11	7	4	3	3	0
20	20	20	0	8	7	8	1	7	2	2	0
25	25	25	0	10	5	9	4	10	9	9	0
30	28	29	1	12	6	6	0	14	7	7	0
35	27	**26**	-1	14	9	9	0	18	2	3	1
40	29	30	1	16	4	7	3	26	7	7	0
45	23	24	1	18	3	9	6	30	10	10	0
50	20	**19**	-1	20	2	8	6	34	8	8	0
instances_10_a50_s2-10_l20-30_L50-100				instances_10_a50_s2-10_l20-30_L250-300				instances_10_a50_s20-30_l3-10_L50-100			
1	4	4	0	5	9	9	0	7	23	23	0
6	8	8	0	10	5	6	1	11	27	27	0
11	7	**7**	0	15	8	8	0	15	20	20	0
16	4	5	1	20	8	8	0	19	23	23	0
21	9	9	0	25	3	5	2	23	23	24	1
26	5	5	0	30	7	7	0	27	28	30	2
31	5	5	0	35	7	**7**	0	31	27	27	0
36	4	4	0	40	6	6	0	35	23	24	1
41	2	2	0	45	7	7	0	39	26	28	2
46	10	10	0	50	6	6	0	43	24	**23**	-1
instances_10_a50_s20-30_l3-10_L250-300				instances_10_a50_s20-30_l20-30_L50-100				instances_10_a50_s20-30_l20-30_L250-300			
2	21	21	0	4	26	**17**	-9	4	30	32	2
7	21	21	0	9	21	**20**	-1	9	23	23	0
12	26	26	0	14	21	**18**	-3	14	20	20	0
17	28	28	0	19	27	**16**	-11	19	22	23	1
22	29	29	0	24	22	**17**	-5	24	24	24	0
27	27	27	0	29	30	**21**	-9	29	27	**26**	-1
32	28	28	0	34	24	**18**	-6	34	22	22	0
37	30	30	0	39	22	**19**	-3	39	20	20	0
42	26	26	0	44	23	**18**	-5	44	22	22	0
47	25	25	0	49	30	**19**	-11	49	29	29	0

TABLE 6.8: Comparison Between Lagrange [18] and HAS approach for 100 target strings. "instance no" represents the instance indices we have run, "Lagrange" means results of that instance by Lagrangian relaxation approach [18], "HAS" represents same instance run by our approach. "Diff" represents the difference between Lagrangian relaxation approach and HAS approach. where $(+n)$ means $Solution_{HAS} = Solution_{Lagrange} + n$ and $(-n)$ means $Solution_{HAS} = Solution_{Lagrange} - n$.

instance no	Lagrange	HAS	Diff	instance no	Lagrange	HAS	Diff	instance no	Lagrange	HAS	Diff
instances_100_a4_s2-10_l20-30_L50-100				instances_100_a4_s2-10_l20-30_L250-300				instances_100_a4_s20-30_l20-30_L50-100			
2	8	8	0	1	5	5	0	2	23	25	2
7	4	4	0	6	8	8	0	7	26	26	0
12	5	5	0	11	7	7	0	12	29	36	7
17	8	8	0	16	7	7	0	17	21	23	2
22	2	2	0	21	8	8	0	22	23	26	3
27	7	7	0	26	4	4	0	27	24	27	3
32	5	5	0	30	3	28	25	32	20	20	0
37	7	7	0	36	5	5	0	37	27	28	1
42	8	8	0	41	9	9	0	42	24	25	1
47	5	5	0	46	5	5	0	47	24	24	0
instances_100_a4_s20-30_l20-30_L250-300				instances_100_a20_s2-10_l3-10_L50-100				instances_100_a20_s2-10_l3-10_L250-300			
1	23	23	0	3	2	2	0	1	3	4	1
6	25	27	2	8	7	7	0	6	3	3	0
11	28	30	2	13	6	6	0	11	9	9	0
16	22	22	0	18	3	7	4	16	10	10	0
21	21	21	0	23	10	10	0	21	10	10	0
26	29	32	3	28	6	6	0	26	5	7	2
31	21	21	0	33	7	7	0	31	4	4	0
36	20	20	0	38	8	8	0	36	4	4	0
41	27	27	0	43	10	11	1	41	10	10	0
46	23	23	0	48	5	5	0	46	6	6	0
instances_100_a20_s2-10_l20-30_L50-100				instances_100_a20_s2-10_l20-30_L250-300				instances_100_a20_s20-30_l3-10_L50-100			
1	4	4	0	1	9	9	0	10	24	24	0
6	6	6	0	6	10	10	0	14	29	29	0
11	5	5	0	11	9	9	0	18	23	25	2
16	7	7	0	16	4	4	0	22	27	27	0
21	9	9	0	21	6	6	0	26	21	21	0
26	6	6	0	26	10	10	0	30	20	20	0
31	3	3	0	31	2	2	0	34	24	24	0
36	2	3	1	36	10	10	0	38	20	20	0
41	8	8	0	41	5	5	0	42	28	28	0
46	7	7	0	46	7	7	0	46	29	29	0
instances_100_a20_s20-30_l3-10_L250-300				instances_100_a20_s20-30_l20-30_L50-100				instances_100_a20_s20-30_l20-30_L250-300			
5	23	23	0	2	25	25	0	2	25	25	0
10	21	21	0	7	22	22	0	7	22	22	0
15	28	28	0	12	30	30	0	12	30	30	0
20	30	30	0	17	29	29	0	17	22	22	0
25	29	29	0	22	27	27	0	22	26	26	0
30	20	22	2	27	26	26	0	27	25	25	0
35	25	25	0	32	20	20	0	32	29	29	0
40	29	31	2	37	23	23	0	37	22	22	0
45	20	38	18	42	25	25	0	42	27	27	0
50	21	21	0	47	30	30	0	47	22	22	0
instances_100_a50_s2-10_l3-10_L50-100				instances_100_a50_s2-10_l3-10_L250-300				instances_100_a50_s2-10_l20-30_L50-100			
1	3	6	3	1	23	23	0	3	10	10	0
2	9	9	0	6	25	27	2	8	10	10	0
4	5	6	1	11	28	30	2	13	3	3	0
7	6	6	0	16	22	22	0	18	3	3	0
11	2	2	0	21	21	21	0	23	6	6	0
14	8	9	1	26	29	32	3	28	10	10	0
16	9	9	0	31	21	21	0	33	5	5	0
19	6	6	0	36	20	20	0	38	8	8	0
21	9	9	0	41	27	27	0	43	8	8	0
23	8	8	0	46	23	23	0	48	2	2	0
instances_100_a50_s2-10_l20-30_L250-300				instances_100_a50_s20-30_l3-10_L50-100				instances_100_a50_s20-30_l3-10_L250-300			
1	7	7	0	7	23	23	0	2	21	21	0
6	9	9	0	11	26	26	0	7	20	20	0
10	8	8	0	15	26	26	0	12	28	28	0
16	7	7	0	19	21	21	0	17	22	24	2
21	6	6	0	23	26	26	0	22	24	24	0
26	4	4	0	27	24	24	0	27	27	27	0
31	10	10	0	31	22	22	0	32	27	27	0
36	10	10	0	35	21	21	0	37	27	27	0
41	10	10	0	39	21	21	0	42	20	20	0
46	3	10	7	43	25	25	0	47	29	29	0
instances_100_a50_s20-30_l20-30_L50-100				instances_100_a50_s20-30_l20-30_L250-300							
1	22	22	0	2	23	23	0				
6	29	29	0	7	26	26	0				
11	20	20	0	12	26	26	0				
16	26	26	0	17	21	21	0				
21	21	21	0	22	20	20	0				
26	26	26	0	27	26	26	0				
31	24	24	0	32	25	25	0				
36	29	29	0	37	27	27	0				
41	28	29	1	42	27	27	0				
46	30	30	0	47	21	21	0				

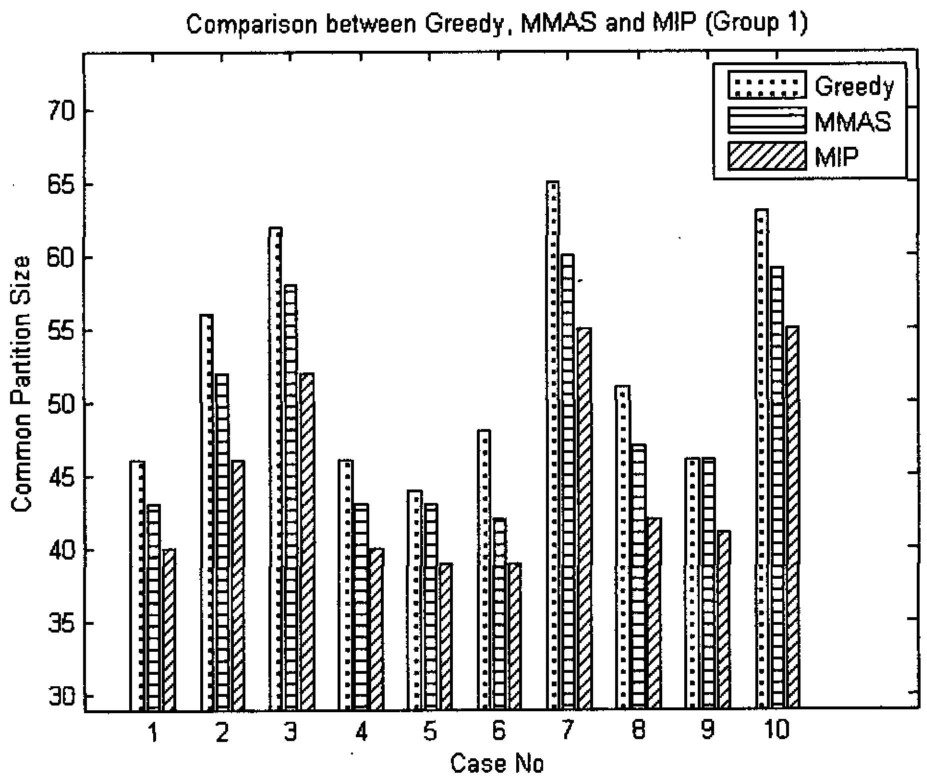

FIGURE 6.1: Comparison between greedy, MMAS and MIP(Group 1)

Table 6.12, 6.13 and 6.14 present the results of student t-test. The first 3 columns summarize the t-statistic result for greedy vs. MMAS. The first column reports the t-value of two sample t-test. A positive t-value indicate significant improvement. The second column presents the p-value. A lower p-value represent higher significant improvement and the third column reports whether the null hypothesis is rejected or accepted. Here the null hypothesis is that the two random population (partition sizes from greedy and MMAS) have equal means. We have used $+, -, \approx$ to denote improvement, deteriotion and almost equal respectively. From the table, we can see that out of 30 instances our approach gets better partition size for 28 cases. According to t-statistic value with 5% significance value we have found better solution in 26 cases for MMAS. The other 3 cases shows no improvement and for one case we got worse result in 5% significance level. The fourth, fifth and sixth column present the t-statistic result for MMAS vs. MIP. the t-value, p-value and the significance represent the same meaning as before but now they are calculated for the MIP approach.

Figure 6.1, 6.2 and 6.3 show the bar plots of partition sizes achieved by greedy [20], MMAS and MIP approach.

TABLE 6.9: Comparison between Greedy approach [20], MMAS and MIP on random DNA sequences (Group 1, 200 bps). Here, Difference = MMAS(Avg.) - Greedy, Difference1 = MIP - Greedy and Difference2 = MIP - MMAS(Avg.). Best and Worst report the maximum and minimum partition size among 15 runs using MMAS.

greedy	MMAS(Avg.)	Worst	Best	Difference	Std.Dev.(MMAS	Time in sec(MMAS)	MIP	Difference1	Difference2
46	42.8667	43	42	-3.1333	0.3519	114.6243	40	-6	-2.8667
56	51.8667	52	51	-4.1333	0.5164	100.823	46	-10	-5.8667
62	57	58	55	-5	0.6547	207.5253	52	-10	-5
46	43.3333	43	43	-2.6667	0.488	168.3098	40	-6	-3.3333
44	42.9333	43	43	-1.0667	0.2582	42.7058	39	-5	-3.9333
48	42.8	43	42	-5.2	0.414	75.2033	39	-9	-3.8
65	60.6	60	60	-4.4	0.5071	131.9478	55	-10	-5.6
51	46.9333	47	47	-4.0667	0.4577	201.2292	42	-9	-4.9333
46	45.5333	46	45	-0.4667	0.5164	172.6809	41	-5	-4.5333
63	59.7333	60	59	-3.2667	0.7037	288.4226	55	-8	-4.7333

TABLE 6.10: Comparison between Greedy approach [20], MMAS and MIP on random DNA sequences (Group 2, 400 bps). Here, Difference = MMAS(Avg.) - Greedy, Difference1 = MIP - Greedy and Difference2 = MIP - MMAS(Avg.). Best and Worst report the maximum and minimum partition size among 15 runs using MMAS

greedy	MMAS(Avg.)	Worst	Best	Difference	Std.Dev.(MMAS	Time in sec(MMAS)	MIP	Difference1	Difference2
119	113.9333	116	111	-5.0667	1.3345	1534.1015	101	-18	-12.9333
122	118.9333	121	117	-3.0667	0.9612	1683.1146	108	-14	-10.9333
114	112.5333	114	111	-1.4667	0.8338	1398.5315	101	-13	-11.5333
116	116.4	117	115	0.4	0.7368	1739.3478	104	-12	-12.4
135	132.2	135	130	-2.8	1.3202	1814.7264	117	-18	-15.2
108	106.0667	107	105	-1.9333	0.8837	1480.2378	99	-9	-7.0667
108	98.4	101	96	-9.6	1.2421	1295.2485	95	-13	-3.4
123	118.4	120	117	-4.6	0.7368	1125.2353	109	-14	-9.4
124	119.4667	121	117	-4.5333	1.0601	1044.4141	107	-17	-12.4667
105	101.8667	103	101	-3.1333	0.7432	1360.1529	91	-14	-10.8667

112495

TABLE 6.11: Comparison between greedy approach [20], MMAS and MIP on random DNA sequences (Group 3, 600 bps). Here, Difference = MMAS(Avg.) - greedy, Difference1 = MIP - greedy and Difference2 = MIP - MMAS(Avg.). Best and Worst report the maximum and minimum partition size among 15 runs using MMAS

greedy	MMAS(Avg.)	Worst	Best	Difference	Std.Dev.(MMAS	Time in sec(MMAS)	MIP	Difference1	Difference2
182	180	181	177	-2	2	1773.0398	159	-23	-21
175	176.25	177	175	1.25	0.9574	3966.8293	162	-13	-14.25
196	188	189	187	-8	0.8165	1589.2953	172	-24	-16
192	184.25	185	184	-7.75	0.5	2431.158	-1	—	—
176	171.75	173	171	-4.25	0.9574	1224.8943	154	-22	-17.75
170	163.25	165	160	-6.75	2.2174	1826.1438	153	-17	-10.25
173	168.5	170	167	-4.5	1.291	1802.1655	151	-22	-17.5
185	176.25	177	175	-8.75	0.9574	1838.5603	157	-28	-19.25
174	172.75	175	172	-1.25	1.5	4897.4688	157	-17	-15.75
171	167.25	168	167	-3.75	0.5	1886.2098	155	-16	-12.25

TABLE 6.12: t-statistic summary result for greedy vs. MMMAS and MMAS vs. MIP (Group 1)

tstat	p-value	significance	tstat	p-value	significance
34.4886	0.0000	+	31.5534	0.0000	+
31	0.0000	+	44	0.0000	+
29.5804	0.0000	+	29.5804	0.0000	+
21.166	0.0000	+	26.4575	0.0000	+
16	0.0000	+	59	0.0000	+
48.6415	0.0000	+	35.5457	0.0000	+
33.6056	0.0000	+	42.7707	0.0000	+
34.4086	0.0000	+	41.7416	0.0000	+
3.5	0.0016	+	34	0.0000	+
17.9781	0.0000	+	26.0499	0.0000	+

TABLE 6.13: t-statistic summary result for greedy vs. MMMAS and MMAS vs. MIP (Group 2)

tstat	p-value	significance	tstat	p-value	significance
14.7042	0.0000	+	37.5344	0.0000	+
12.3572	0.0000	+	44.0562	0.0000	+
6.8126	0.0000	+	53.5715	0.0000	+
-2.1026	0.0446	-	65.1815	0.0000	+
8.2143	0.0000	+	44.5921	0.0000	+
8.4731	0.0000	+	30.9705	0.0000	+
29.9333	0.0000	+	10.6014	0.0000	+
24.1802	0.0000	+	49.4118	0.0000	+
16.5622	0.0000	+	45.5459	0.0000	+
16.328	0.0000	+	56.6269	0.0000	+

TABLE 6.14: t-statistic summary result for greedy vs. MMMAS and MMAS vs. MIP (Group 3)

tstat	p-value	significance	tstat	p-value	significance
2	0.0924	$\simeq$	21	0	+
-2.6112	0.0401	$\simeq$	29.7673	0	+
19.5959	0	+	39.1918	0	+
31	0	+	—	—	—
8.878	0.0001	+	37.0785	0	+
6.0883	0.0009	+	9.2452	0.0001	+
6.9714	0.0004	+	27.1109	0	+
18.2782	0	+	40.2119	0	+
1.6667	0.1466	$\simeq$	21	0	+
15	0	+	49	0	+

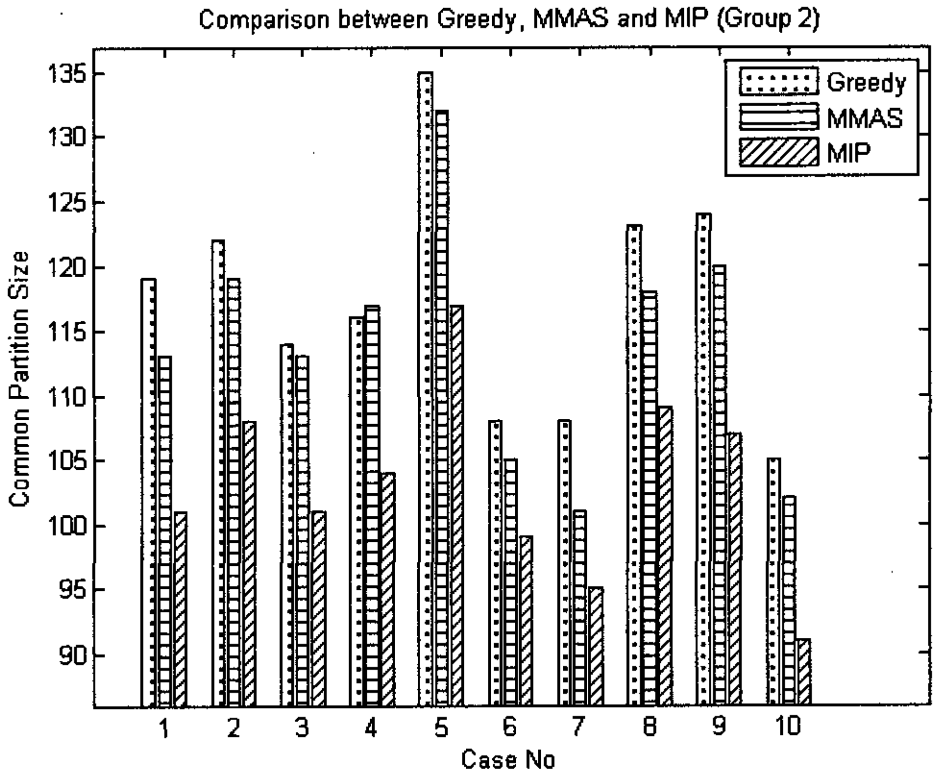

FIGURE 6.2: Comparison between greedy, MMAS and MIP (Group 2)

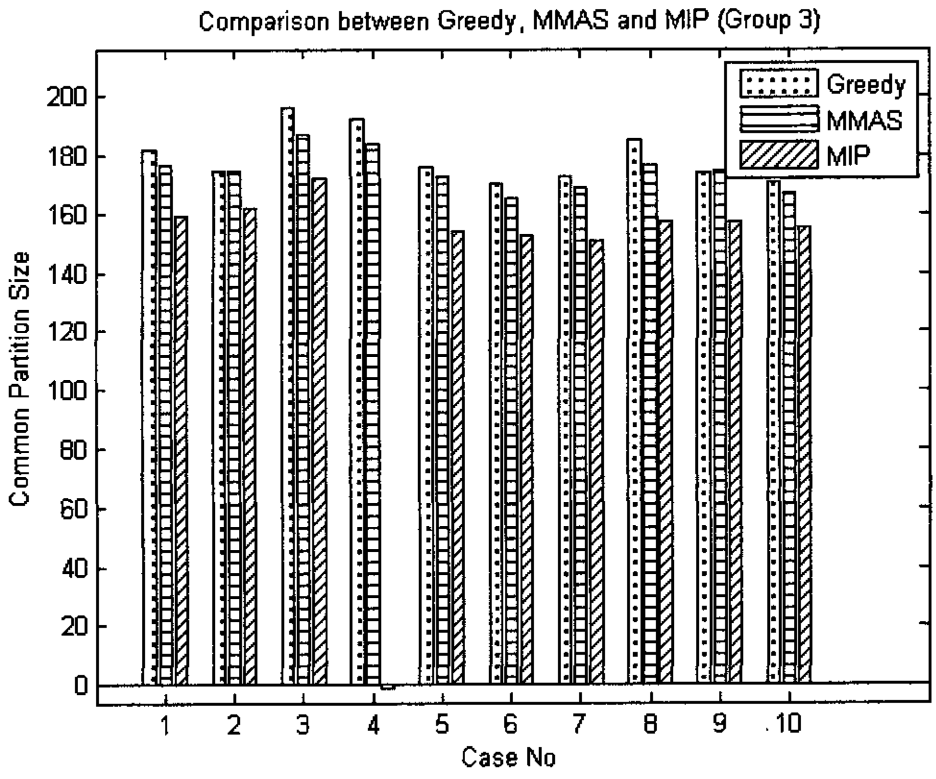

FIGURE 6.3: Comparison between greedy, MMAS and MIP (Group 3)

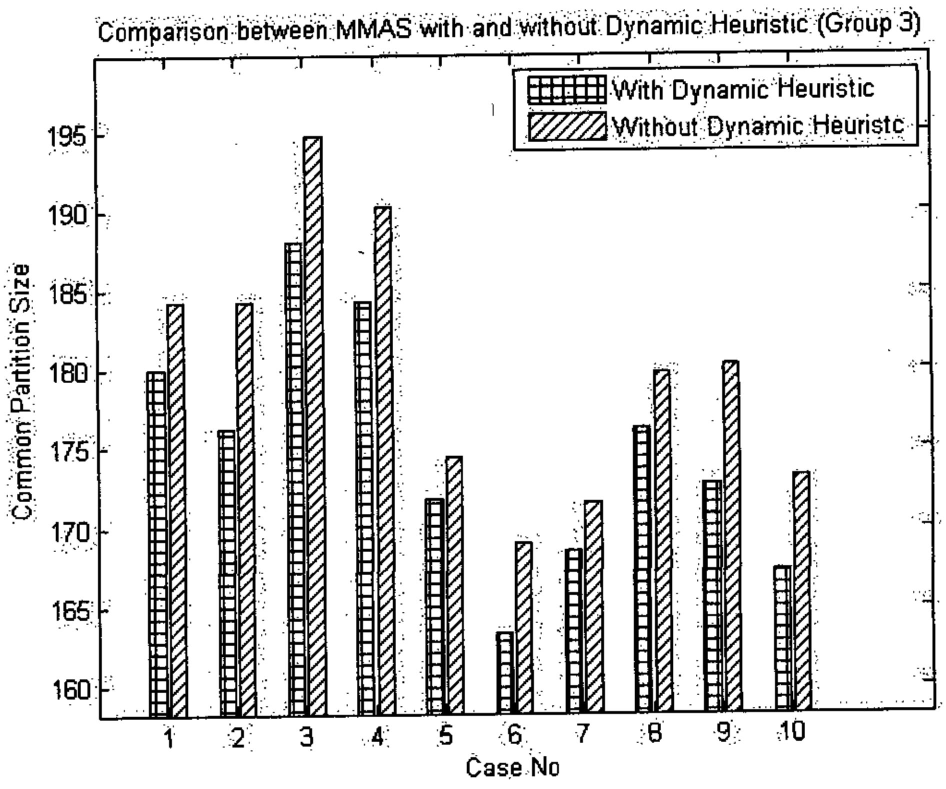

FIGURE 6.4: Comparison between MMAS with and without dynamic heuristic (Group 1)

6.4.2.2 Effects of Dynamic Heuristics

In Chapter 5 (Section 5.2.2.2), we discussed the dynamic heuristic we employ in our algorithm. We conducted experiments to check and verify the effect of this dynamic heuristic. We conducted experiments with two versions of our algorithm- with and without applying the dynamic heuristic. The effect is presented in Table 6.15, where for each group the average partition size with dynamic heuristic and without dynamic heuristic is reported. The positive difference depicts the improvement using dynamic heuristic. Out of 30 cases we found positive differences on 27 cases. This clearly shows the significant improvement using dynamic heuristics. It can also be observed that with the increase in length, the positive differences are increased. Figures 6.4, 6.5, and 6.6 show the case by case results.

TABLE 6.15: Comparison between MMAS with and without dynamic heuristic on random dna sequence

Group 1 (200 bps)			Group 2 (400 bps)			Group 3 (600 bps)		
MMAS	**MMAS(w/o heuristic)**	**Difference**	**MMAS**	**MMAS(w/o heuristic)**	**Difference**	**MMAS**	**MMAS(w/o heuristic)**	**Difference**
42.7500	43.2500	0.5000	114.2500	115.5000	1.2500	180.0000	183.2500	3.2500
51.5000	50.7500	-0.7500	119.0000	121.0000	2.0000	176.2500	183.2500	7.0000
56.7500	56.5000	-0.2500	112.2500	113.5000	1.2500	188.0000	193.7500	5.7500
43.0000	44.0000	1.0000	116.2500	120.5000	4.2500	184.2500	189.2500	5.0000
43.0000	42.7500	-0.2500	132.2500	134.0000	1.7500	171.7500	173.5000	1.7500
42.2500	42.5000	0.2500	105.5000	107.7500	2.2500	163.2500	168.0000	4.7500
60.0000	60.5000	0.5000	99.0000	99.7500	0.7500	168.5000	170.5000	2.0000
47.0000	47.5000	0.5000	118.0000	121.7500	3.7500	176.2500	178.7500	2.5000
45.7500	46.0000	0.2500	119.5000	120.7500	1.2500	172.7500	179.2500	6.5000
59.2500	61.5000	2.2500	101.7500	103.7500	2.0000	167.2500	172.2500	5.0000

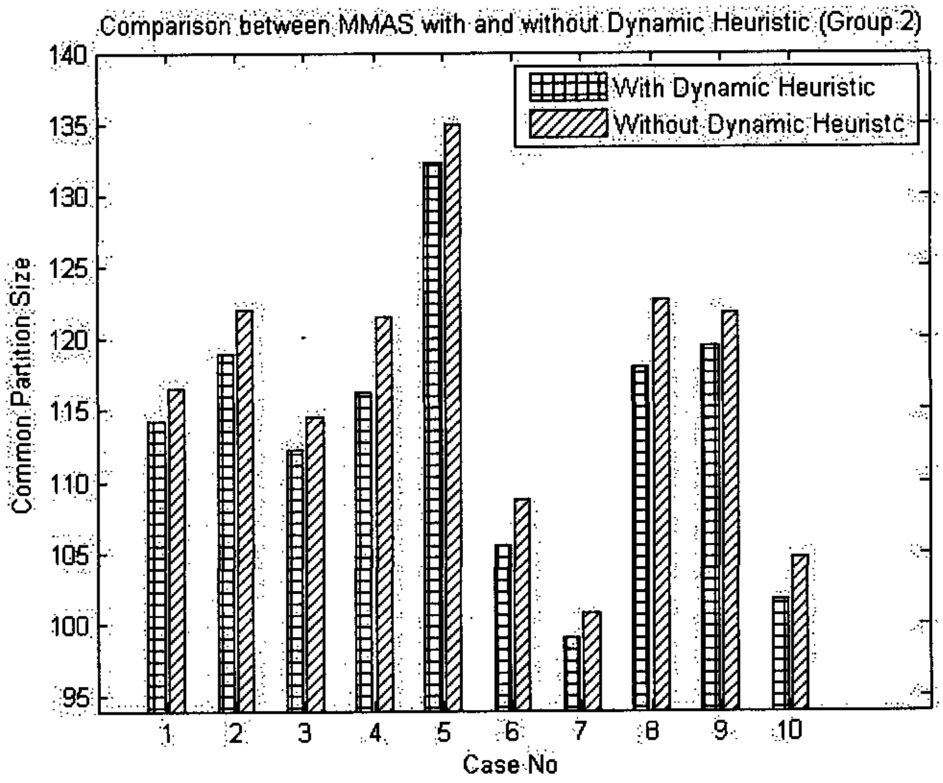

FIGURE 6.5: Comparison between MMAS with and without dynamic heuristic (Group 2)

6.4.2.3 Real Gene Sequence

Table 6.16 shows the minimum common partition size found by our approach and the greedy approach for the real gene sequences. Out of the 15 instances we get better results on 11 instances. The t-statistic result shows almost equal result on 5 cases and improvement in the other 10 cases in 5% significance level.

Figure 6.7 shows the bar plots of partition sizes achieved by Greedy [20], MMAS and MIP approach for real gene sequence.

6.4.2.4 Runtime Analysis

The greedy solution runs very fast. In our experiment we found that the greedy gives output within 2 minutes. As the problem is an offline problem, we consider the running time is not significant here. The metaheuristic approaches like ACO requires greater time than greedy approaches. We omitted the runtime analysis of MMAS vs. greedy. But we are interested in running time analysis of MIP vs.

TABLE 6.16: Comparison between greedy approach [20], MMAS and MIP on real gene sequence. Here, Difference = MMAS(Avg.) - greedy, Difference1 = MIP - greedy and Difference2 = MIP - MMAS(Avg.). Best and Worst report the maximum and minimum partition size among 15 runs using MMAS

greedy	MMAS	Worst	Best	Difference	Std.Dev(MMAS)	Time in sec(MMAS)	MIP	Difference1	Difference2
95	87.66666667	88	87	-7.333333333	0.487950036	863.8083333	79	-16	-8.666666667
161	156.3333333	162	154	-4.666666667	2.350278606	1748.34	140	-21	-16.33333333
121	117.0666667	118	116	-3.933333333	0.883715102	1823.4922	110	-11	-7.066666667
173	164.8666667	167	163	-8.133333333	1.187233679	1823.012533	145	-28	-19.86666667
172	173.2	175	171	1.2	1.207121724	2210.153533	157	-15	-16.2
153	146	148	143	-7	1.309307341	1953.838267	134	-19	-12
140	141	142	140	1	0.755928946	2439.0346	128	-12	-13
134	133.1333333	136	130	-0.866666667	1.807392228	1406.804533	122	-12	-11.13333333
149	147.5333333	150	145	-1.466666667	1.505545305	2547.519267	134	-15	-13.53333333
151	150.5333333	152	148	-0.466666667	1.597617273	1619.6364	132	-19	-18.53333333
126	125	127	123	-1	1	1873.3868	116	-10	-9
143	139.1333333	141	137	-3.866666667	1.245945806	2473.249067	130	-13	-9.133333333
180	181.5333333	184	179	1.533333333	1.35576371	2931.665333	165	-15	-16.53333333
152	149.3333333	151	147	-2.666666667	1.290994449	2224.403733	136	-16	-13.33333333
157	161.6	164	160	4.6	1.242118007	1739.612133	145	-12	-16.6

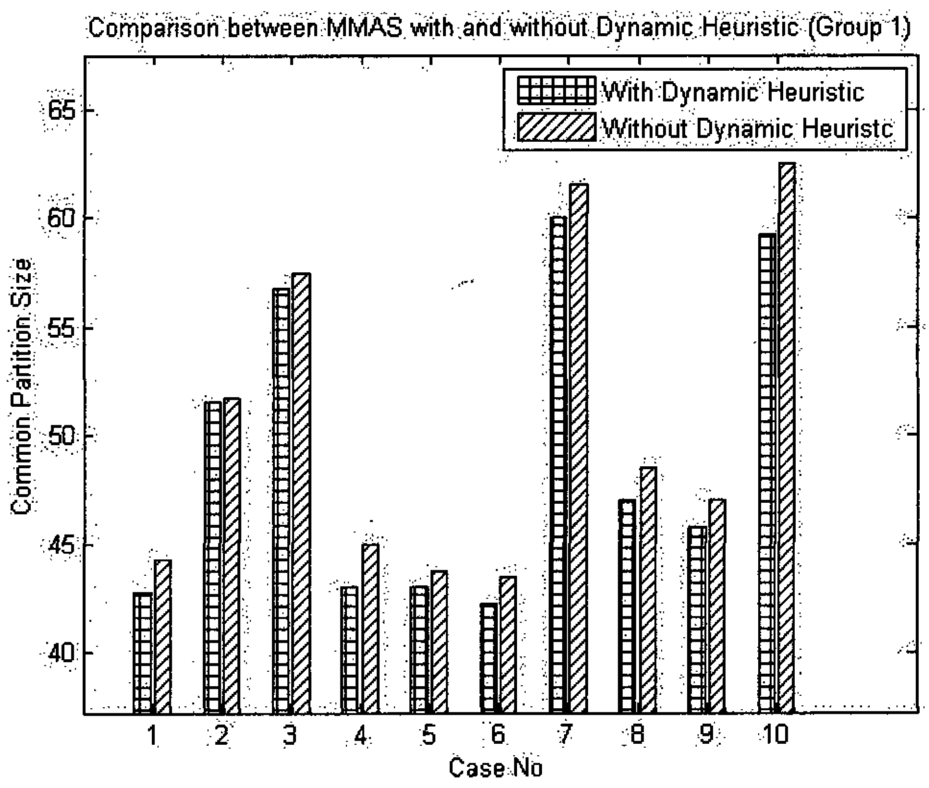

FIGURE 6.6: Comparison between MMAS with and without dynamic heuristic (Group 3)

TABLE 6.17: t-statistic summary result for Greedy Vs. MMMAS and MMAS Vs. MIP (Real Gene Seqn.)

tstat	p-value	significance	tstat	p-value	significance
58.2065	0.0000	+	68.7895	0.0000	+
7.6901	0.0000	+	26.9154	0.0000	+
17.2383	0.0000	+	30.9705	0.0000	+
26.5325	0.0000	+	64.8089	0.0000	+
-3.8501	0.0006	+	51.9768	0.0000	+
20.7063	0.0000	+	35.4965	0.0000	+
-5.1235	0.0000	-	66.6052	0.0000	+
1.8571	0.0738	≈	23.8571	0.0000	+
3.7730	0.0008	+	34.8142	0.0000	+
1.1313	0.2675	≈	44.9290	0.0000	+
3.8730	0.0006	+	34.8569	0.0000	+
12.0194	0.0000	+	28.3907	0.0000	+
-4.3802	0.0002	-	47.2304	0.0000	+
8.0000	0.0000	+	40.0000	0.0000	+
-14.3430	0.0000	-	51.7596	0.0000	+

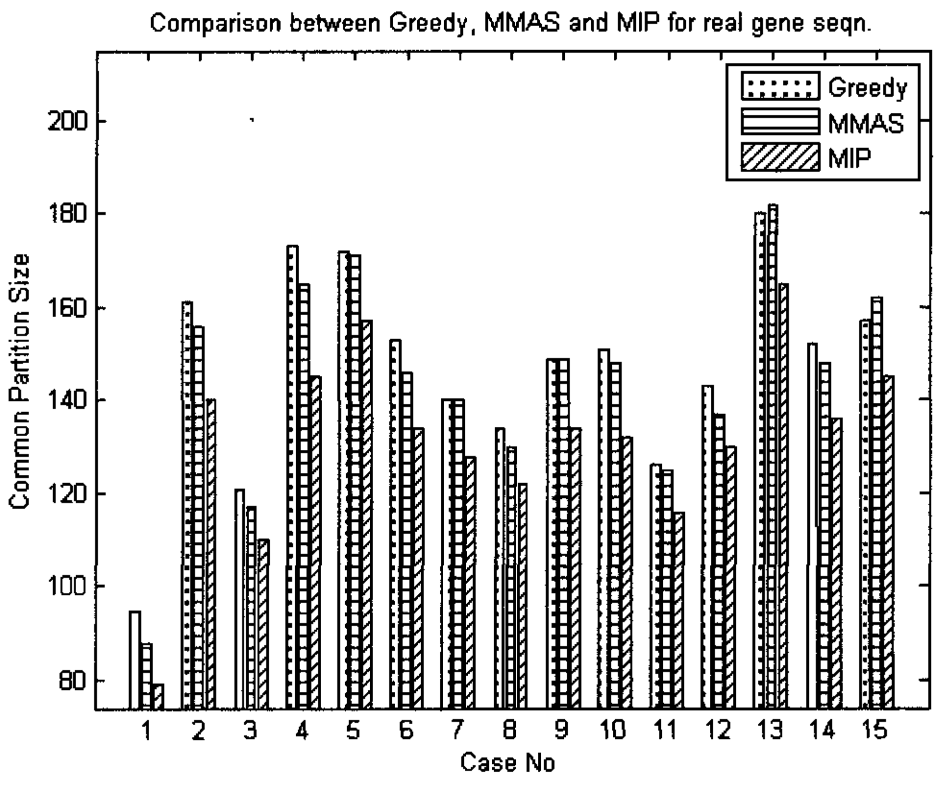

FIGURE 6.7: Comparison between greedy, MMAS and MIP(Real Gene Seqn.)

greedy. We experimented with the group 1 random dna sequences. The MIP is run given 30s, 60s, 120s and 300s as time limit. The common partition sizes are compared with the greedy solution in Figure 6.8.

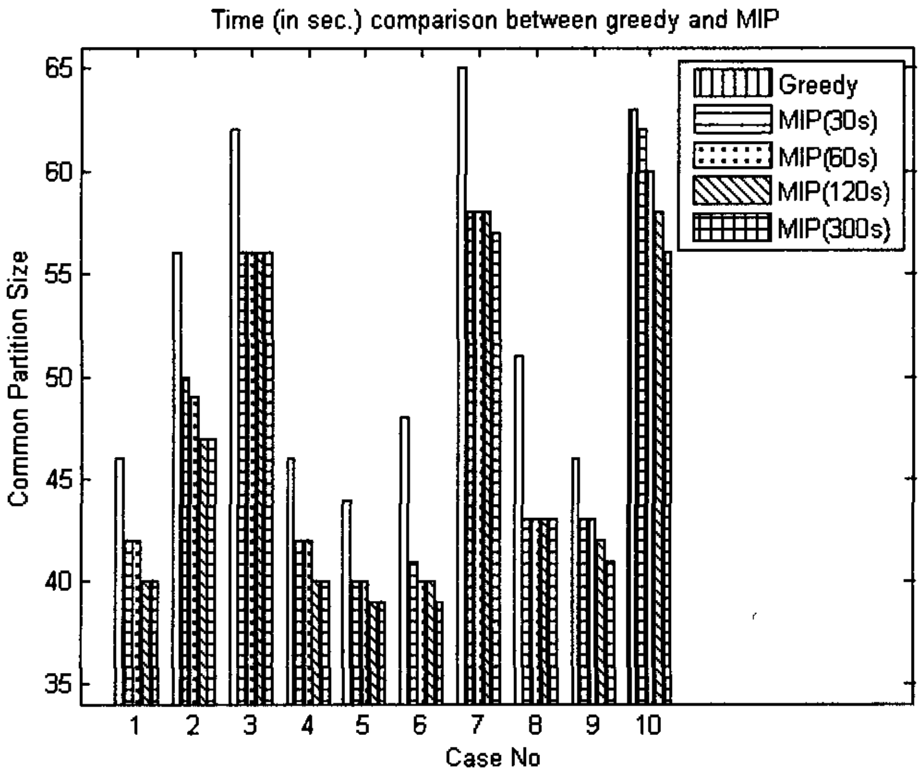

FIGURE 6.8: Time (in sec.) comparison between greedy and MIP

Chapter 7

Conclusion

In this chapter, we draw the conclusion by highlighting the major contributions made by the research works associated with this thesis. We also provide some directions for future research on the problems handled here.

7.1 Major Contributions

The contributions that have been made in this thesis can be enumerated as follows:

- We have devised an efficient graph mapping from the problem instances. The graphs are *substring graph* and *common substring graph.*
- For MCSP problem, we have developed an efficient MILP formulation using the *common substring graph.*
- Some dynamic heuristics have also been developed for MCSP problem.
- Using the graphs and the heuristic, we have also implemented Hybrid Ant System (HAS) and MAX-MIN Ant System (MMAS) for MSC and MCSP problems respectively.
- We have done extensive experiment and compared our results with the state of the art algorithms in literature.

7.2 Future Directions of Further Research

A number of future directions can be given. Below we will list the future directions and further research in these two problems.

- We have applied two variants of ACO namely HAS for MSC problem and MMAS for MCSP problem. A good future research could be to apply other variants like ACS to solve these two problems.
- As there are not many string problems solved by ACO in the literature, a good future research could be to apply our developed model to solve other hard string problems.
- In this thesis we have applied ACO metaheuristics to solve MSC and MCSP problems. In future it might be interesting to develop other metaheuristics such as Genetic Programming, Artificial Immune system, Simulated Annealing etc. to solve these two problems.

Bibliography

[1] Gnu linear programming kit, version 4.48.

[2] J. Abara. Applying integer linear programming to the fleet assignment problem. *Interfaces*, 19(4):pp. 20–28, 1989.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[4] D. Applegate and W. J. Cook. A computational study of the job-shop scheduling problem. *INFORMS Journal on Computing*, 3(2):149–156, 1991.

[5] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics).* Princeton University Press, Princeton, NJ, USA, 2007.

[6] V. Bafna, Pavel, and A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 1998.

[7] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM J. Comput.*, 25(2):272–289, Feb. 1996.

[8] F. Bahredar, H. Erfani, H. Javadi, and N. Masaeli. A meta heuristic solution for closest string problem using ant colony system. In A. Leon F. de Carvalho, S. Rodríguez-González, J. Paz Santana, and J. Rodríguez, editors, *Distributed Computing and Artificial Intelligence*, volume 79 of *Advances in Intelligent and Soft Computing*, pages 549–557. Springer Berlin Heidelberg, 2010.

[9] E. Balas and M. W. Padberg. On the set-covering problem. *Operations Research*, 20(6):1152–1161, 1972.

[10] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Oper. Res.*, 36(3):493–513, May 1988.

[11] M. Besten, T. Stützle, and M. Dorigo. Ant colony optimization for the total weighted tardiness problem. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VI*, volume 1917 of *Lecture Notes in Computer Science*, pages 611–620. Springer Berlin Heidelberg, 2000.

[12] C. Blum. Beam-aco: Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Comput. Oper. Res.*, 32(6):1565–1591, June 2005.

[13] C. Blum. Beam-aco for the longest common subsequence problem. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[14] C. Blum and M. J. Blesa. New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Comput. Oper. Res.*, 32(6):1355–1377, June 2005.

[15] C. Blum, M. Y. Vallès, and M. J. Blesa. An ant colony optimization algorithm for dna sequencing by hybridization. *Comput. Oper. Res.*, 35(11):3620–3635, Nov. 2008.

[16] H. Bodlaender, R. G. Downey, M. R. Fellows, M. T. Hallett, and H. T. Wareham. Parameterized complexity analysis in computational biology. *Comput. Appl. Biosci*, 11:49–57, 1995.

[17] R. Borndörfer and R. Weismantel. Set packing relaxations of some integer programs. *Mathematical Programming*, 88(3):425–450, 2000.

[18] S. Canzar, T. Marschall, S. Rahmann, and C. Schwiegelshohn. Solving the minimum string cover problem. In D. A. Bader and P. Mutzel, editors, *ALENEX*, pages 75–83. SIAM / Omnipress, 2012.

[19] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(4):302–315, Oct. 2005.

[20] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *ACM Trans. Algorithms*, 1(2):350–366, Oct. 2005.

[21] D. Costa and A. Hertz. Ants can colour graphs. *The Journal of the Operational Research Society*, 48(3):295–305, 1997.

[22] P. Damaschke. Minimum common string partition parameterized. In K. Crandall and J. Lagergren, editors, *Algorithms in Bioinformatics*, volume 5251 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin Heidelberg, 2008.

[23] L. M. de Campos, J. M. Fernández-Luna, J. A. Gámez, and J. M. Puerta. Ant colony optimization for learning bayesian networks. *International Journal of Approximate Reasoning*, 31(3):291 – 311, 2002.

[24] M. Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, Italy, 1992.

[25] M. Dorigo and G. D. Caro. The ant colony optimization meta-heuristic. In *in New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[26] M. Dorigo, A. Colorni, and V. Maniezzo. Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1991.

[27] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artif. Life*, 5(2):137–172, Apr. 1999.

[28] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Trans. Evol. Comp*, 1(1):53–66, Apr. 1997.

[29] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B*, 26(1):29–41, 1996.

[30] M. Dorigo and T. Stützle. Ant colony optimization: Overview and recent advances. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 227–263. Springer US, 2010.

[31] M. Dorigo and T. Stützle. *Ant Colony Optimization.* Bradford Company, Scituate, MA, USA, 2004.

[32] R. L. Dorit and W. Gilbert. The limited universe of exons. *Current Opinion in Genetics & Development*, 1(4):464 – 469, 1991.

[33] Z. Drezner and H. Hamacher. *Facility location. Applications and Theory.* Springer, Berlin, 2002.

[34] J. Edmonds and E. Johnson. Matching, euler tours and the chinese postman. *Mathematical Programming*, 5(1):88–124, 1973.

[35] S. Faro and E. Pappalardo. Ant-csp: An ant colony optimization algorithm for the closest string problem. In J. Leeuwen, A. Muscholl, D. Peleg, J. Pokorný, and B. Rumpe, editors, *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *Lecture Notes in Computer Science*, pages 370–381. Springer Berlin Heidelberg, 2010.

[36] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements.* The MIT Press, 1st edition, 2009.

[37] M. L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Manage. Sci.*, 50(12 Supplement):1861–1871, Dec. 2004.

[38] G. Fuellerer, K. F. Doerner, R. F. Hartl, and M. Iori. Ant colony optimization for the two-dimensional loading vehicle routing problem. *Comput. Oper. Res.*, 36(3):655–673, Mar. 2009.

[39] L. Gambardella and M. Dorigo. Ant-q: A reinforcement learning approach to the traveling salesman problem. pages 252–260. Morgan Kaufmann, 1995.

[40] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS J. on Computing*, 12(3):237–255, July 2000.

[41] L. M. Gambardella, E. Taillard, and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. Technical report, 1999.

[42] A. Geoffrion. Lagrangian relaxation for integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 243–281. Springer Berlin Heidelberg, 2010.

[43] N. Giorgetti. Glpkmex, a matlab mex interface for the glpk library.

[44] B. Golden, S. Raghavan, and E. A. Wasil. *The vehicle routing problem : latest advances and new challenges.* Operations research/Computer science interfaces series, 43. Springer, 2008.

[45] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partitioning problem: Hardness and approximations. *The Electronic Journal of Combinatorics*, 12(R50), 2005.

[46] R. E. Gomory. Outline of an algorithm for integer solutions to linear program. *Bulletin of the American Mathematical Society*, 64(5):275–278, September 1958.

[47] M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):pp. 1138–1162, 1970.

[48] D. Hermelin, D. Rawitz, R. Rizzi, and S. Vialette. The minimum substring cover problem. In C. Kaklamanis and M. Skutella, editors, *Approximation and Online Algorithms*, volume 4927 of *Lecture Notes in Computer Science*, pages 170–183. Springer Berlin Heidelberg, 2008.

[49] H. Hernández and C. Blum. Ant colony optimization for multicasting in static wireless ad-hoc networks. *Swarm Intelligence*, 3(2):125–148, 2009.

[50] K. Hoffman and M. Padberg. Lp-based combinatorial problem solving. *Annals of Operations Research*, 4(1):145–194, 1985.

[51] K. Hoffman and T. Ralphs. Integer and combinatorial optimization. In S. Gass and M. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 771–783. Springer US, 2013.

[52] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Manage. Sci.*, 39(6):657–682, June 1993.

[53] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[54] H. Jiang, B. Zhu, D. Zhu, and H. Zhu. Minimum common string partition revisited. In *Proceedings of the 4th International Conference on*

Frontiers in Algorithmics, FAW'10, pages 45–52, Berlin, Heidelberg, 2010. Springer-Verlag.

[55] D. Jungnickel. *Graphen, Netzwerke und Algorithmen*. BI-Wissenschaftsverlag, 1990.

[56] A. Land and A. Doig. An automatic method for solving discrete programming problems. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer Berlin Heidelberg, 2010.

[57] L. Lessing, I. Dumitrescu, and T. Stützle. A comparison between aco algorithms for the set covering problem. In M. Dorigo, M. Birattari, C. Blum, L. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence*, volume 3172 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2004.

[58] M. López-Ibáñez, C. Blum, D. Thiruvady, A. Ernst, and B. Meyer. Beam-aco based on stochastic sampling for makespan optimization concerning the tsp with time windows. In C. Cotta and P. Cowling, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 5482 of *Lecture Notes in Computer Science*, pages 97–108. Springer Berlin Heidelberg, 2009.

[59] V. Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS J. on Computing*, 11(4):358–369, Apr. 1999.

[60] V. Maniezzo and A. Carbonaro. An ant heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16:927–935, 1999.

[61] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[62] D. Martens, M. De Backer, R. Haesen, J. Vanthienen, M. Snoeck, and B. Baesens. Classification with ant colony optimization. *Trans. Evol. Comp*, 11(5):651–665, Oct. 2007.

[63] D. Merkle and M. Middendorf. Ant colony optimization with global pheromone evaluation for scheduling a single machine. *Applied Intelligence*, 18(1):105–111, 2003.

[64] D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. In *IEEE Transactions on Evolutionary Computation*, pages 893–900. Morgan Kaufmann, 2000.

[65] J. Neraud. Elementariness of a finite set of words is co-np-complete. *ITA*, 24:459–470, 1990.

[66] F. E. Otero, A. A. Freitas, and C. G. Johnson. cant-miner: An ant colony classification algorithm to cope with continuous attributes. In *Proceedings of the 6th International Conference on Ant Colony Optimization and Swarm Intelligence*, ANTS '08, pages 48–59, Berlin, Heidelberg, 2008. Springer-Verlag.

[67] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.*, 33(1):60–100, Feb. 1991.

[68] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas. Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6:321–332, 2002.

[69] B. H. Partee, A. ter Meulen, and R. E. Wall. *Mathematical Methods in Linguistics*, volume 30 of *Studies in Linguistics and Philosophy*. Kluwer, Dordrecht, 1990.

[70] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer Publishing Company, Incorporated, 3rd edition, 2008.

[71] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426 – 438, 2004. Financial Risk in Open Economies.

[72] M. Reimann, K. Doerner, and R. F. Hartl. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Comput. Oper. Res.*, 31(4):563–591, Apr. 2004.

[73] F. Rendl. Semidefinite relaxations for integer programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 687–726. Springer Berlin Heidelberg, 2010.

[74] R. Shah. Optimization problems in sonet/wdm ring architecture, 1998.

[75] S. J. Shyu and C.-Y. Tsai. Finding the longest common subsequence for multiple biological sequences by ant colony optimization. *Comput. Oper. Res.*, 36(1):73–91, Jan. 2009.

[76] K. Socha and C. Blum. An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing and Applications*, 16(3):235–247, 2007.

[77] K. Socha, J. Knowles, and M. Sampels. A max-min ant system for the university course timetabling problem. In M. Dorigo, G. Caro, and M. Sampels, editors, *Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2002.

[78] K. Socha, M. Sampels, and M. Manfrin. Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In S. Cagnoni, C. Johnson, J. Cardalda, E. Marchiori, D. Corne, J.-A. Meyer, J. Gottlieb, M. Middendorf, A. Guillot, G. Raidl, and E. Hart, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 334–345. Springer Berlin Heidelberg, 2003.

[79] C. Solnon. Combining two pheromone structures for solving the car sequencing problem with ant colony optimization.

[80] C. Solnon and S. Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, May 2006.

[81] P. Stothard. The sequence manipulation suite: Javascript programs for analyzing and formatting protein and dna sequences. *Biotechniques*, 28(6):1102, 2000.

[82] T. Stützle and H. Hoos. Improving the ant system: A detailed report on the max-min ant system. Technical report, 1996.

[83] T. Stützle and H. Hoos. Max-min ant system and local search for the traveling salesman problem. In *IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION (ICEC'97)*, pages 309–314. IEEE Press, 1997.

[84] T. Stützle and H. H. Hoos. Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914, June 2000.

[85] T. Stützle, F. Intellektik, F. Informatik, and T. H. Darmstadt. An ant approach to the flow shop problem. In *In Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98*, pages 1560–1564. Verlag, 1997.

[86] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Airline crew scheduling: A new formulation and decomposition algorithm. *Operations Research*, 45:188–200, 1995.

[87] P. Villesen. Fabox: An online fasta sequence toolbox, 2007.

[88] G. Watterson, W. Ewens, T. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1 – 7, 1982.