DNA Sequencing by Hybridization: Optimization with Ant Colonies

(Seqüenciació d'ADN per hibridació: optimització amb colònies de formigues)

Mateu Yábar Vallès

Essay submited in partial fulfilment of the requirements for the degree of Superior Engineer in Informatics to the Facultat de Informàtica de Barcelona at the Universitat Politècnica de Catalunya

Advisors: María Blesa and Christian Blum

Academic year 2005-2006

Agraïments

Primerament m'agradaria agraïr el gran esforç que han fet tant el Christian Blum com la Maria Blesa per ajudar-me, en tot moment, a dur a terme aquest projecte. També al Jordi Petit, per haver-me guiat en els moments inicials.
D'altra banda m'agradaria agraïr als meus pares l'ajuda prestada. També a tots els habitants de Can Xandri, haver-me cedit el balcó com a despatx. Tampoc vull oblidar les formigues d'una platja asturiana, que van patir els meus experiments. Finalment, voldria agraïr a tothom que ha hagut d'escoltar la meva història sobre formigues i ADN quatre o cinc vegades, mai no haver protestat.
A tots, merci.

Contents

List of Algorithms

Chapter 1

Planificació i contribucions d'aquest PFC

Aquesta memòria és el resultat d'un projecte de fi de carrera per a la Facultat d'Informàtica de Barcelona, de la Universitat Politècnica de Catalunya. És un projecte de recerca l'objectiu del qual és desenvolupar algorismes que permetin solucionar el problema de la seqüenciació d'ADN per hibridació d'una forma més eficaç que les que s'han proposat fins ara.

En aquest capítol, primer explicarem breument l'estructura de la memòria. A continuació, exposarem la planificació inicial del projecte, i comentarem les variacions que hi ha hagut. Detallarem el cost econòmic del projecte i, finalment, exposarem les contribucions científiques que aporta.

1.1 Estructura de la memòria

La memòria es composa de tres parts. La primera, que conté el gruix més important del treball és el desenvolupament del projecte en sí. Hi expliquem el problema que volem resoldre i com el resoldrem, i fem un estudi complet de diferents tàctiques per tractar-lo des d'un punt de vista algorísmic. Als paràgrafs següents introduïm, amb més detall, el contingut de cada capítol. La segona part de la memòria consisteix en un annex amb tot de resultats dels nostres algorismes. La tercera part és un CD-Rom que conté el codi font dels algorismes explicats a la memòria i la bateria d'instàncies del problema que hem utilitzat com a joc de proves en els experiments realitzats amb els nostres algorismes.

Excepte aquest capítol introductori, la resta de la memòria està redactada en anglès. El marcat caràcter de recerca d'aquest projecte, així com la importància de transmetre a la comunitat científica els coneixements i resultats que s'han pogut extreure dels estudis realitzats, ens han fet triar aquesta llengua per escriure la part més científica de la memòria. Això també ens ha permès optimitzar l'escriptura d'aquesta memòria amb la escriptura simultània de dos articles de recerca.

Capítol 2

En el segon capítol fem una introducció al problema de la seqüenciació per hibridació (SBH), que serà el problema que tractarem en aquest projecte, així com a les eines algorísmiques que farem servir per tractar-lo. Primer fem una descripció de l'orígen i fonaments biològics

del problema i, posteriorment, el definim formalment i el formulem com a problema (computacional) d'optimització. Explicarem algunes de les tècniques que s'utilitzen per tractar problemes d'optimització, i finalment farem una breu descripció bibliogràfica de les aproximacions algorísmiques existents que han intentat tractar el problema de SBH.

Capítol 3

Així com en el capítol anterior hem explicat el problema a resoldre, en aquest capítol proposarem heurístiques constructives per tractar-lo i, a continuació, farem estudis experimentals per estudiar la qualitat d'aquestes heurístiques.

Capítol 4

En aquest capítol explicarem en detall els fonaments de la metaheurística *Ant Colony Optimization* (ACO), que permet optimitzar problemes combinatòrics basant-se en el comportament biològic de les colònies de formigues que trobem a la natura. Començarem explicant els fonaments biològics d'aquesta metaheurística, que es basen en el rastre que deixen les formigues mentre caminen i que les permet retornar al formiguer i alhora comunicar a d'altres formigues del mateix formiguer on trobar menjar. De manera natural, aquest comportament local permet establir camins mínims entre el formiguer i les fonts de menjar, optimitzant-ne així aquesta activitat. Veurem que aquest comportament natural es pot descriure algorísmicament i per tant transformar en codi que podem executar. El capítol es tanca amb la descripció d'algunes de les variants més importants de la metaheurística ACO.

Capítol 5

En el capítol 2 hem explicat el problema i en el 3 hem creat creant uns primers algorísmes; en el capítol 4 hem explicat la meta-heurística ACO. En aquest capítol farem una explicació molt detallada de com hem aplicat ACO en el problema de seqüenciació d'ADN per hibrididació. Primer presentarem dos algorismes basats en dues variacions de l'ACO original ($\mathcal{MM}$AS i ACS). A continuació, introduirem un entorn algorísmic multinivell que podrem aplicar als algorismes ACO descrits anteriorment per manipular les instàncies inicials.

Capítol 6

En la primera part del capítol 6, farem proves experimentals sobre els algorismes ACO descrits en el capítol anterior, per tal de fixar el valor dels paràmetres que regulen el comportament de l'algorisme. Un cop decidits quins són els millors valors pels paràmetres, analitzarem experimentalment d'una forma exhaustiva el comportament dels algorismes proposats, i farem un estudi comparatiu detallat dels resultats.

Capítol 7

En aquest darrer capítol, explicarem breument les conclusions que hem extret del treball descrit en els capítols anteriors.

1.2 Planificació i cost de la feina

La idea inicial del projecte va nèixer a començament del curs 2005–2006, a partir d'unes quantes propostes que em van fer els meus directors, en Christian Blum i la Maria Blesa. Entre les opcions de problemes d'optimització a tractar que em van proposar, vaig decidir escollir-ne el SBH per diverses raons; les principals foren, per una banda, la seva simplicitat de descripció, la relació clara amb el món real, i d'altra banda la possible aplicabilitat dels resultats que es poguessin obtenir.

El primer quadrimestre del curs 2005–2006 va servir per escollir el problema a tractar i per concretar-ne els objectius i el procediment a seguir en el seu estudi. Just en començar el segon quadrimestre, vam plantejar una planificació orientativa de la feina a fer fins al juny. El gruix del treball d'aquest projecte s'ha dut a terme en aquest segon quadrimestre.

En tots els estadis de la realització del projecte, s'han realitzat nombroses reunions (pràcticament setmanals) amb els directors per debatre i decidir diferents aspectes del treball. Aquesta freqüència de contactes s'ha intensificat encara més cap al final del projecte. D'aquesta manera, he estat molt ben guiat, de cara a aconseguir els objectius marcats.

Les Taules 1.1 i 1.2 detallen les principals tasques realitzades des de febrer del present any fins al moment d'enquadernació d'aquesta memòria.

La realització de les tasques inicialment planificades, com s'especifica a la taules 1.1 i 1.2, ha patit lleugeres modificacions. Com ja hem dit anteriorment, el projecte va començar a prendre forma en el primer quatrimestre del curs 2005-2006. Aquesta posada en marxa va ser lenta, per les dificultats que hi va haver a l'hora de trobar un problema adient, i per la poca disponibilitat horària. Per aquesta raó, el treball efectiu en aquest projecte va començar realment a partir de Febrer; és per això, que la planificació de les tasques comença a partir d'aquest mes.

Cost econòmic del projecte

El cost econòmic d'aquest projecte no és gaire alt. Calcular-ho, però, no és fàcil, ja que en un projecte de recerca és difícil posar preu al temps invertit en pensar, provar coses que després no funcionen i no s'inclouen al producte final, el temps invertit en llegir bibliografia, en formació autodidàctica d'eines que es van necessitant sobre la marxa, etc.

Per obtenir un cost aproximat d'aquest projecte, ens hem basat en el sou d'un becari FPI del ministeri d'educació i ciència. Actualment, aquest tipus de becari predoctoral té un sou brut de 1100 €. Segons això, si considerem que per realitzar aquest treball s'hi requereixen 5 mesos a dedicació completa, el cost de la dedicació humana requerida ascendeix a 5500 €.

En aquest cost de 5500 €hem d'afegir el cost del material i recursos necessaris. Soposant que es disposa d'un lloc físic on treballar (és a dir, que no comptarem despeses de mobiliari ni de lloguer d'espai, ni d'utilització d'energia elèctrica, ni de material de papereria), es requereix almenys un ordinador personal per al desenvolupament d'aquest projecte. Això afegeix uns 1500 €al presupost necessari per a la feina humana.

Així doncs, el cost aproximat d'aquest projecte seria d'uns **5500 €**, si només es comptabilitza el treball humà, o seria d'uns **7000 €**), si s'hagués d'incloure la compra d'un computador.

Òbviament, considerar només l'utilització d'una màquina és la configuració de sistema més simple. Si, tal i com hem fet en el projecte, en lloc d'utilitzar només una màquina volguéssim

fer servir un cluster de computadors, llavors el cost del projecte s'incrementaria molt. Només per tenir una idea del cost que suposaria això, el cluster *nozomi* del departament de LSI que hem fet servir en els nostres experiments, va costar al voltant de 15000 €el passat any 2005.

1.3 Contribucions d'aquest PFC

En el món científic actual, el pes que està tenint la genètica en els nous avanços mèdics és molt gran i això fa que tots els problemes relacionats amb el tractament d'ADN adquireixin una gran importància. La seqüenciació d'ADN per hibridització no n'és una excepció.

Val a senyalar la importància dels resultats obtinguts mitjançant les heurístiques constructives proposades, que superen amb escreix les que hi ha descrites en la bibliografia actual. Volem remarcar que la descripció i el testeig de les heurístiques seran publicades en el congrés WABI 2006, 6th Workshop on Algorithms in Bioinformatics, a Zurich. Les actes del congrés WABI 2006 seran publicades per l'editorial Springer-Verlag, sota les seves conegudes sèries Lecture Notes in Computer Science (LNCS).

S'han de remarcar també en els resultats obtinguts per les heurístiques ACO. Tot i que aquestes heurístiques no aconsegueixen superar la millor heurística proposada, aconsegueixen igualar-la. Remarquem que la proposta dels diferents algorismes basats en ACO que hem proposat ha estat enviada en forma d'article al congrés HM 2006, 3rd International Workshop on Hybrid Metaheuristics, a Gran Canària, i a data d'avui, quan escric aquesta memòria, encara està pendent d'acceptació.

A nivell més personal, m'agradaria dir que en fer aquest projecte he pogut aprendre moltes coses. La primera, i la més important: he après la metodologia que s'ha de seguir per desenvolupar projectes d'investigació i en particular he aprés, com desenvolupar metaheurístiques per tractar problemes d'optimització. En el mateix sentit, m'he familiaritzat amb la lectura de treballs científics (en anglès), i he adquirit l'hàbit de la reflecció posterior a la lectura. Voldria remarcar que, en aquest procés, he après a ser crític amb els treballs publicats, cosa que m'ha fet ser el màxim de rigurós a l'hora d'exposar els meus resultats. Totes aquestes coses no vaig tenir l'oportunitat d'aprendre-les amb les assignatures que conformen la carrera.

D'altra banda, he adquirit coneixements sobre la utilització d'eines que em poden ser molt útils de cara al futur, començant per un domini del llenguatge de programació C++ i de la seva llibreria STL. A l'hora de redactar de la memòria, he hagut d'aprendre LaTeX, per tal d'editar textos i crear imatges, i a fer servir els programes de distribució lliure Gnuplot i R per a tractar estadísticament i representar gràficament els resultats. També he millorat els meus coneixements del llenguatge Python, que he usat per desenvolupar scripts per al tractament i automàtic dels resultats. Degut que els algorismes han estat executats en un cluster (nozomi) del departament de LSI de la UPC, he hagut d'aprendre a executar programes en una màquina remota mitjançant els coneixements necessaris del sistema operatiu Linux.

Finalment, el fet de redactar la memòria en anglès m'ha fet millorar la meva expresió en aquesta llengua. Inicialment va ser força difícil, però a mesura que avançava l'escriptura de la memòria, la meva redacció ha anat millorant. En aquest sentit, cal destacar que, degut a que paral·lelament a la memòria s'han escrit articles per conferències, s'han fet moltes correccions sobre la redacció.

Temps planificat	Tasca
Febrer 2006	Familiarització amb el problema a tractar. Cerca bibliogràfica i lectura d'aquesta bibliografia que tracta el problema. Lectura dels treballs existents en aplicació de metaheurístiques al problema. Paral·lelament, lectura de textos bàsics en optimització amb colònies de formigues (que es preveu utilitzar al final). Es decideix el conjunt d'instàncies que servirà de joc de proves dels algorismes.
Març 2006	Primeres propostes d'heurístiques per resoldre de manera aproximada el problema. S'en proposen sis, de les quals només s'inclouen en aquest treball quatre (LAG, SH, FB-LAG i FB-SH). Les altres dues es descarten pel seu mal funcionament sobre les instàncies que es proven. Es realitzen experiments exhaustius per testejar la qualitat i el comportament d'aquestes heurístiques proposades. Paral·lelament es comença a escriure el capítol introductori de la memòria. Això suposa haver de familiaritzar-se amb LaTeX, un llenguatge per produir/processar text desconeguda per mi.

Table 1.1: Distribució de tasques per l'hivern 2005–2006

Temps planificat	Tasca
Abril 2006	Es proposen tres noves heurístiques (SM, HSM i S-HSM) que semblen inicialment molt prometedores. Es proven diverses variacions d'aquestes que no resulten en millores respecte a les tres mencionades. Es comencen a realitzar experiments més exhaustius per determinar la qualitat i comportament d'aquestes heurístiques, i comparar-les amb les altres heurístiques proposades. Els experiments ens permeten decidir quines heurístiques utilitzar en la nostra aproximació al problema i incloure en aquesta memòria. El tractament de les dades i la creació de gràfiques fa que m'hagi de familiaritzar amb els programes Gnuplot i R, que permeten la creació de gràfiques i el tractament estadístic de les dades. Es comença a redactar el capítol de descripció de totes les heurístiques. Per començar amb les metaheurístiques, es reimplementa una de les existents a la bibliografia (un *greedy randomized adaptive search procedure* o GRASP). Es troba que els autors fan trampa en el seu article i, com que implementant l'algorisme adequadament, els resultats no són bons respecte a les heurístiques, es descarta aquesta aproximació.
Maig 2006	Es realitzen les primeres propostes per aplicar metaheurístiques basades en colònies de formigues. La dificultat de programar l'algorisme allarga una mica aquest període. Finalment, es proposa un primer algorisme (el $\mathcal{MM}$AS) basat en colònies de formigues per tractar el problema. Es descobreix que, curiosament, és important que una part de la colònia de formigues intenti completar la seqüenciació d'esquerra a dreta i una altra part de dreta a esquerra. Es proposa també una segona metaheurística pel problema (el ACS) basada en formigues. La forma de procedir d'aquest últim és diferent a l'anterior, ja que són variants diferents del que es coneix com *ant colony optimization* o ACO. Es fan alguns experiments inicials. S'escriu un capítol de la memòria introduïnt ACO i es comença l'escriptura de la descripció dels algorismes proposats.
Juny 2006	Es proposa un nou algorisme basat en ACO que incorpora la idea de resolució del problema a diferents nivells, mitjançant la compressió (o reducció) del problema, la resolució d'un problema de menor talla, i la posterior extensió d'aquesta solució al nivell del problema inicial. D'això se'n diu un entorn algorísmic multinivell. Es fan experiments exhaustius i detallats per estudiar-ne el comportament i qualitat de solucions dels algorismes dissenyats al maig i d'aquest últim algorisme basat en resolució per nivells. S'acaba la redacció de la memòria. La segona quinzena del mes i els 4 primers dies de juliol es dedicaran a fer la presentació.

Table 1.2: Distribució de tasques per a la primavera 2006

Chapter 2

Introduction: the problem and the existing approaches

In this chapter we first present the problem that will be studied in this document, introducing both practical and computational aspects. Afterwards we present the state of the art algorithms to tackle the problem.

2.1 The biological point of view

Deoxyribonucleic acid (DNA) is a molecule that contains the genetic instructions for the biological development of all cellular forms of life. Each DNA molecule consists of two (complementary) sequences of four different nucleotide bases, namely adenine (A), cytosine (C), guanine (G), and thymine (T). In mathematical terms each of these sequences can be represented as a word from the alphabet $\{A, C, G, T\}$. One of the most important problems in computational biology consists in determining the exact structure of a DNA molecule, called *DNA sequencing*. This is not an easy task, because the nucleotide base sequence of a DNA molecule (henceforth also called DNA strands) are usually too large to be read in one piece. In 1977, 24 years after the discovery of DNA, two separate methods for DNA sequencing were developed: the chain termination method and the chemical degradation method. Later, in the late 1980's, an alternative and much faster method called *DNA sequencing by hybridization* was developed (see [1, 28, 22]).

DNA sequencing by hybridization works roughly as follows. The first phase of the method consists of a chemical experiment which requires a so-called DNA array. A DNA array is a two-dimensional grid whose cells typically contain all possible DNA strands—called probes—of equal length l. The set of all probes in a DNA array is denominated a *library*. For example, consider a DNA array of all possible probes of length $l = 3$:

GGT	**TGA**	**GCG**	**CTA**	**AAT**	**CCT**	**CTC**	**TTC**
GTC	**GTG**	**TTG**	**GGC**	**CGA**	**TTT**	**TCA**	**ATC**
GCT	**AGC**	**GGG**	**CCA**	**TAT**	**CGG**	**TAG**	**AAG**
GAA	**GGA**	**CGT**	**ACG**	**CTG**	**TGT**	**TAA**	**ATT**
TCT	**GAT**	**CAC**	**CAT**	**CAA**	**ACC**	**ATG**	**GTT**
CGC	**GCC**	**AGG**	**CTT**	**ATA**	**TCC**	**TGC**	**GTA**
AGA	**AAC**	**TTA**	**TGG**	**TCG**	**AGT**	**CAG**	**GAC**
CCG	**GCA**	**CCC**	**AAA**	**ACA**	**GAG**	**ACT**	**TAC**

GGT	**TGA**	**GCG**	**CTA**	**AAT**	**CCT**	**CTC**	**TTC**
GTC	**GTG**	**TTG**	**GGC**	**CGA**	**TTT**	**TCA**	**ATC**
GCT	**AGC**	**GGG**	**CCA**	**TAT**	**CGG**	**TAG**	**AAG**
GAA	**GGA**	**CGT**	**ACG**	**CTG**	**TGT**	**TAA**	**ATT**
TCT	**GAT**	**CAC**	**CAT**	**CAA**	**ACC**	**ATG**	**GTT**
CGC	**GCC**	**AGG**	**CTT**	**ATA**	**TCC**	**TGC**	**GTA**
AGA	**AAC**	**TTA**	**TGG**	**TCG**	**AGT**	**CAG**	**GAC**
CCG	**GCA**	**CCC**	**AAA**	**ACA**	**GAG**	**ACT**	**TAC**

Figure 2.1: Perfect hybridization experiment (i.e, free of errors) involving the target sequence ACTGACTC and all probes of size $l = 3$

After the generation of the DNA array, the chemical experiment is started. It consists of bringing together the DNA array with many copies of the DNA sequence to be read, also called the DNA *target sequence*. Hereby, the target sequence might react with a probe on the DNA array if and only if the probe is a subsequence of the target sequence. Such a reaction is called hybridization. After the experiment, the DNA array allows the identification of the probes that reacted with target sequences. This subset of probes is called the *spectrum*.

An illustration of the hybridization experiment involving the target sequence ACTGACTC and a library of length $l = 3$ is depicted in Figure 2.1. The highlighted cells are those corresponding to the spectrum (i.e., which have reacted during the hybridization experiment).

The second phase of the sequencing by hybridization technique consists in using the spectrum to determine the unknown DNA target sequence. The reconstruction of the original sequence consists of finding an order of the spectrum elements in which each pair of neighboring elements overlaps on $l-1$ letters (i.e., the last $l-1$ letters of each probe coincide with the first $l-1$ letters of the next).

However, the hybridization experiment usually produces errors in the spectrum. There are two types of errors:

1. **Negative errors:** Some probes that should be in the spectrum (because they appear in the target sequence) do not appear in the spectrum. A particular type of negative error is caused by the multiple existence of a probe in the target sequence. This cannot be detected by the hybridization experiment (i.e., the library can only detect the presence of oligonucleotides, not their amount). Such a probe will appear at most once in the spectrum. In the reconstruction, the presence of negative errors forces overlapping between some neighboring oligonucleotides in a sequence on fewer than $l-1$ letters.

2. **Positive errors:** A probe of the spectrum that does not appear in the target sequence is called a positive error. In the second phase, the presence of positive errors forces some oligonucleotides to be rejected during the reconstruction process.

2.2 The computational problem

The computational part of the hybridization experiment is to achieve the reconstruction of the target sequence with the oligonucleotides obtained in the first phase (i.e., the spectrum). In the case that the obtained spectrum is perfect (that is, free of errors), the original sequence

can be reconstructed in polynomial time with an algorithm proposed by Pevzner in [31]. The existence of errors in the spectrum results in strongly NP-hard combinatorial problems, as shown by Błażewicz and Kasprzak in [7].

In order to solve the computational part of DNA sequencing by hybridization, one usually solves an optimization problem of which the optimal solutions can be shown to have a high probability to resemble the target sequence. In this work we consider the optimization problem that was introduced as a model for DNA sequencing by hybridization by Błażewicz et al. in [3].

2.2.1 The model

Instance: A set S (spectrum) of words of equal length l over the alphabet {A,C,G,T} (i.e., $S = \{\{A, C, G, T\}^l\}^*$) and the length n of the original DNA target sequence.
Goal: To find a sequence of length $\leq n$ containing the maximal number of elements of S.

The mathematical-programming formulation of the problem is given below.

$$\text{Maximize} \quad \sum_{i=1}^{z}\sum_{j=1}^{z} b_{ij} + 1 \tag{2.1}$$

$$\text{subject to} \quad \sum_{i=1}^{z} b_{ik} \leq 1, \quad k = 1, \ldots, z \tag{2.2}$$

$$\sum_{i=1}^{z} b_{ki} \leq 1, \quad k = 1, \ldots, z \tag{2.3}$$

$$\sum_{k=1}^{z} \left(\left| \sum_{i=1}^{z} b_{ki} - \sum_{j=1}^{z} b_{jk} \right| \right) = 2 \tag{2.4}$$

$$\sum_{s_k \in S'} \left(\sum_{s_i \in S'} b_{ik} \cdot \sum_{s_j \in S'} b_{kj} \right) < |S'|, \quad \forall S' \subset S, S' \neq \emptyset \tag{2.5}$$

$$\sum_{i=1}^{z}\sum_{j=1}^{z} w_{ij} b_{ij} \leq n - l \tag{2.6}$$

where:

S is the spectrum,

$s_i \in S$ is an element of the spectrum,

$z = |S|$ is the cardinality of the spectrum,

n is the length of the original DNA target sequence,

l is the length of each spectrum element,

b_{ij} is a binary variable equal to 1 if the element s_i is the immediate predecessor of the element s_j in a solution; otherwise it is equal to 0,

w_{ij} = is the cost of a connection of element s_i with element s_j equal to the difference between l and a number of letters of the common part of s_i and s_j coming from their maximal overlapping.

The *objective function* (2.1) counts the number of spectrum elements composing the solution. Inequalities (2.2) and (2.3) guarantee that every element of the spectrum will be joined in the solution with at most one element from the right side, respectively one element from the left side. The addition of equation (2.4) ensures that exactly two elements connected from only one side with other elements will appear in the solution. These elements will constitute the beginning and the end of the reconstructed DNA target sequence. Supplying the above formulation with (2.5) allows to eliminate the solutions including subcycles of elements. According to (2.6) the length of the reconstructed sequence cannot exceed the length of the DNA target sequence (the length can be shorter, for example, in case of negative errors appearing at the end of the sequence).

Deficiencies of the model

The model described in this section is currently the only one that models the computational part of the sequencing by hybridization problem. However we have found that this model has some deficiencies when representing the original problem. In the following we expose—without a formal demonstration—two situations where the model does not correspond correctly to the original problem. However, due to the focus of this document on practical results we will not propose alternative models to represent the problem in a better way. However we encourage further studies.

The first case that is not well represented by the model appears when the generated sequence is a disordered join of big subsequences that are also subsequences of the target sequence. We will expose this with an example: let s_t be the DNA target sequence which is composed by two subsequences s_{t_0} and s_{t_1} (i.e., $s_t = s_{t_0} + s_{t_1}$). Let $s_c = s_{t_1} + s_{t_0}$ be a solution to the model, that is, a reconstructed target sequence. When s_t is compared with s_c (with methods such as the Needleman-Wunsch algorithm [29] or the Smith-Waterman algorithm [32]) the similarity of both sequences is probably small. However, the model might evaluate s_c as a very good solution. Therefore solutions with high score in the model may be very different from the original DNA target sequence.

In a second scenario the model evaluates with bad results sequences that may be equal or similar to the DNA target sequence. This occurs when the generated solution contains few oligonucleotides but the sequence obtained from it is similar (or equal) to the DNA target sequence. This is because a target sequence s_t can be generated from a solution of only $\left\lceil \frac{|s_t|}{l} \right\rceil$ oligonucleotides. In this case the model would rate the solution as a bad solution when its similarity to the target sequence is high. For a better understanding we propose an imaginary scenario where this situation occurs. Let us imagine a reduced spectrum with many negative errors and no positive errors. The solutions generated from this spectrum will be rated as bad solution with the model but they might be very similar or equal to the target sequence.

2.2.2 Formalization of the problem as a graph problem

The model described can also be studied as a graph problem. In most of this work this new definition is used. However, we give both descriptions in order to simplify explanations. In the following sections a minimal knowledge of graph theory is assumed.

Let s_t be the DNA target sequence of the problem. Let n denote the number of nucleotide bases of s_t. Therefore, s_t can be formally defined as $s_t \in \{A, C, G, T\}^n$ (i.e., as a word of length n over the alphabet $\Sigma = \{A, C, G, T\}$). Furthermore, the spectrum —as obtained by the hybridization experiment— is denoted by S. Let l denote the length of each oligonucleotide (i.e., a short DNA strand) and s be a oligonucleotide which can be formally defined as $s \in \{A, C, G, T\}^l$ and the spectrum $S \in \{\{A, C, G, T\}^l\}^*$. In general, the length of any oligonucleotide s is denoted by $l(s)$.

Let G=(V,E) be a fully connected directed graph defined by $V = S$.[1] Each edge $e = \{s, s'\} \in E$ of the graph has a weight $w_{s,s'} = l - o_{s,s'}$, where $o_{s,s'}$ is the size of the largest sequence that is both a suffix of s and a prefix of s' (i.e., $0 \leq o_{s,s'} < l$). The value $o_{s,s'}$ is the *overlap* between probes s and s'. Let p define a directed path in G. The length of such a path p, denoted by $l(p)$, is defined as the number of vertices (i.e., oligonucleotides) on the path. We define $p[i]$ as the i-th vertex in a given path p (starting from position 1). Therefore $p = \langle p[1] \dots p[l(p)] \rangle$—in this work we will use the notation $\zeta = \langle \gamma_1, \dots, \gamma_n \rangle$ to define that ζ is an ordered set. We also define p_{ini} as the first vertex of p (i.e., $p_{ini} = p[1]$) and p_{end} as the last vertex of p (i.e, $p_{end} = p[l(p)]$). In contrast to the length, the cost of a path p is defined as follows

$$c(p) := l + \sum_{i=1}^{l(p)-1} w_{p[i],p[i+1]} \tag{2.7}$$

The second term is a sum of all the weights of the edges which form part of the path. In fact, $c(p)$ is equivalent to the length of the DNA sequence that is obtained by the sequence of oligonucleotides in p.[2]

According to the model described in [7], the problem of DNA sequencing by hybridization consists in finding a directed path p^* in G with $l(p^*) \geq l(p)$ for all possible paths p that fulfil $c(p) \leq n$, i.e.,

$$p^* = \text{argmax}_{p \in P(G)} \{l(p) | c(p) \leq n\}, \tag{2.8}$$

where $P(G)$ denotes all the possible directed paths in G. In the following we refer to this optimization problem as *sequencing by hybridization* (SBH). Notice that more than one path p^* may be a possible solution of SBH and that in any case, the DNA sequence obtained from p^* may not be equal to the original DNA target sequence s_t.

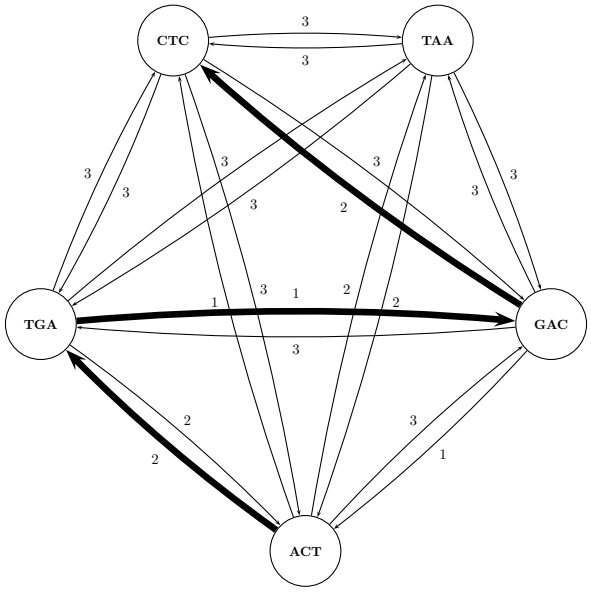
2.2.3 Example

As an example consider the target sequence s_t = ACTGACTC. Assuming $l = 3$, the ideal spectrum is {ACT,CTG,TGA,GAC,ACT,CTC} because it contains all the possible subsequences of length 3 of s_t. However, let us assume that the hybridization experiment provides us with the following faulty spectrum S = {ACT,TGA, GAC,CTC,TAA}. This spectrum has two negative errors, because ACT should appear twice, but can only appear once—due to the characteristics of the hybridization experiment—, and CTG does not appear at all in S. Furthermore, S has one positive error, because it includes oligonucleotide TAA, which does not appear in the target sequence.

An optimal directed path in this example is p^* = ⟨ACT,TGA,GAC,CTC⟩ with $l(p^*) = 4$ and $c(p^*) = 8$. The DNA sequence that is retrieved from this path is equal to the target sequence (see Figure 2.2).

[1] In following sections both notations will be used to refer to the spectrum.

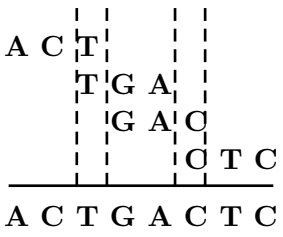
[2] See Section 2.2.3 for an example.



(a) The completely connected directed graph with spectrum S = {ACT,TGA,GAC,CTC,TAA} as the vertex set. The edge weights are also indicated in a table in (b).

	ACT	TGA	GAC	CTC	TAA
ACT	–	2	3	1	2
TGA	2	–	1	3	3
GAC	1	3	–	2	3
CTC	3	3	3	–	3
TAA	2	3	3	3	–

(b) Edge weights of the graph.



(c) DNA sequence retrieval from a directed path.

Figure 2.2: In (a) the optimal path is $p^* = \langle$ACT,TGA,GAC,CTC$\rangle$, which is represented by thicker edges. In (c) is shown how to retrieve the DNA sequence that is encoded by p^*. Note that $c(p^*) = 8$, which is equal to the length of the encoded DNA sequence.

2.3 Combinatorial Optimization Problems

In the previous section we have formulated the SBH problem as a Combinatorial Optimization problem. In general many optimization problems of practical as well as theoretical importance consist of the search for a "best" configuration of a set of variables to achieve some goals. They seem to divide naturally into two categories: those where solutions are encoded with *real-valued* variables, and those where solutions are encoded with *discrete* variables. Among the later ones we find a class of problems called Combinatorial Optimization (CO) problems. According to Papadimitriou and Steigliz [30], in CO problems, we are looking for an object from a finite—or possibly countably infinite—set. This object is typically an integer number, a subset, a permutation or a graph structure. Formally, a *Combinatorial Optimization* problem $P = (\mathcal{S}, f)$ can be defined by:

- a set of variables $X = \{x_1, \ldots, x_n\}$;
- variable domains $D_1, \ldots D_n$;
- constraints among variables;
- an *objective function* f to be maximized[3] where $f : D_1 \times \cdots \times D_n \rightarrow R^+$;

The set of all possible feasible assignments is $\mathcal{S} = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} | v_i \in D_i, s$ satisfies all the constraints $\}$. $\mathcal{S}$ is usually called *search* (or *solution*) *space*, as each element of the set can be seen as a candidate solution. To solve a combinatorial optimization problem one has to find a solution $s^* \in \mathcal{S}$ with maximum objective function value, that is,

$$f(s^*) \geq f(s) \; \forall s \in \mathcal{S}$$

. s^* is called a globally optimal solution of $(\mathcal{S}, f)$ and the set $\mathcal{S}^* \subseteq \mathcal{S}$ is called the set of globally optimal solutions.

Examples of CO problems are the Travelling Salesman problem (TSP), Timetabling and Scheduling problems. As mentioned above the computational part of the DNA sequencing by hybridization is also a CO problem. Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified either as *complete* or *approximate* algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in bounded time. Yet, for CO problems that are NP-hard, no polynomial time algorithm exists, assuming that $\mathsf{P} \neq \mathsf{NP}$. Therefore, complete methods might need exponential time in the worst-case. This often leads to computation times too high for practical purposes. Thus, the use of approximate methods to solve CO problems has received more and more attention in the last 30 years. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time.

Among basic approximate methods we usually distinguish between *constructive* methods and *local search* methods. Constructive algorithms generate solutions from scratch by adding—to an initially empty partial solution—components, until a solution is complete. They are typically the fastest approximate methods, yet they often return solutions of inferior quality when compared to local search algorithms. Local search algorithms start from

[3] As minimizing an objective function f is the same as maximizing $-f$, we will deal, without loss of generality, with maximization problems

some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution, where the neighborhood is formally defined as follows: A *neighborhood structure* is a function $N : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to every solution $s \in \mathcal{S}$ a set of neighbors $N(s) \subseteq \mathcal{S}$. $N(s)$ is called the neighborhood of s. A move is the choice of a solution s' from a neighborhood $N(s)$ of a solution s.

In the last 20 years, a new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called *meta-heuristics*. This class of algorithms includes Ant Colony Optimization (CO), Evolutionary Computation (EC) including Genetic Algorithms (GA), the Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS). Up to now there is no commonly accepted definition for the term meta-heuristic. In short, we could say that meta-heuristics are high level strategies for exploring search spaces by using different methods.

2.3.1 Meta-heuristics

In this section some meta-heuristics are briefly described. For a more extensive overview on meta-heuristics see, for example, [10].

- **Basic Local Search, Iterative Improvement:** Basic local search is usually called *iterative improvement*, since each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum.

- **Simulated Annealing (SA):** The origins of the algorithm are in statistical mechanics (Metropolis algorithm) and it was first presented as a search algorithm for CO problems in Kirkpatrick et al. [27] and Cerny [15]. The fundamental idea is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of doing such a move is decreased during the search.

- **Tabu Search (TS):** Tabu Search is among the most cited and used meta-heuristics for CO problems. The basic idea of TS was first introduced by Glover in [26], based on earlier ideas formulated in Glover [25]. The simple TS algorithm applies a best improvement local search as basic ingredient and uses a *short term memory* to escape from local minima and to avoid cycles. The short term memory is implemented as a *tabu list* that keeps track of the most recently visited solutions and forbids moves towards them. The neighborhood of the current solution is thus restricted to the solutions that do not belong to the tabu list.

- **The Greedy Randomized Adaptive Search Procedure (GRASP):** This technique is a simple meta-heuristic that combines constructive heuristics and local search. GRASP is an iterative procedure, composed of two phases: solution construction and solution improvement. The solution construction mechanism is characterized by two main ingredients: a dynamic constructive heuristic and randomization. Assuming that a solution s consists of a subset of a set of elements (solution components), the solution is constructed step-by-step by adding one new element at a time. The choice of the next element is done by picking it uniformly at random from a candidate list. The elements are ranked by means of a heuristic criterion that gives them a score as a function of the

benefit if inserted in the current partial solution. The candidate list, called *restricted candidate list* (RCL), is composed of the best α elements. The second phase of the algorithm is a local search process, which may be a basic local search algorithm such as iterative improvement, or a more advanced technique such as Simulated Annealing or Tabu Search.

- **Evolutionary Computation (EC):** These algorithms are inspired by nature's capability to evolve living beings well adapted to their environment. EC algorithms can be succinctly characterized as computational models of evolutionary processes. At each iteration a number of operators is applied to the individuals of the current population to generate the individuals of the population of the next generation (iteration). Usually, EC algorithms use operators called *recombination* or *crossover* to recombine two or more individuals to produce new individuals. They also use *mutation* or *modification* operators which cause a self-adaptation of individuals. The driving force in evolutionary algorithms is the *selection* of individuals based on their *fitness* (this can be the value of an objective function or the result of a simulation experiment, or some other kind of quality measure). Individuals with a higher fitness have a higher probability to be chosen as members of the population of the next iteration (or as parent for the generation of new individuals). This corresponds to the principle of *survival of the fittest* in natural evolution. It is the capability of nature to adapt itself to a changing environment, which gave he inspiration for EC algorithms.

- **Ant Colony Optimization (ACO):** Ant Colony Optimization is a meta-heuristic approach proposed in [17, 20, 18]. The inspiring source of ACO is the foraging behavior of real ants. This behavior (as described by Deneubourg et al. in [16]) enables ants to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called pheromone on the ground. When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths. ACO algorithms are based on a parametrized probabilistic model–the pheromone model–that is used to model the chemical pheromone trails. A more detailed description of ACO and its variants can be found in chapter 4.

2.3.2 Existing Approaches to the SBH

In the previous section we introduced some meta-heuristics to tackle Combinatorial Optimization problems such as the SBH problem. However, existing approaches for the SBH problem include complete as well as heuristic and meta-heuristic methods. In this section we will introduce the existing approaches to the SBH problem.

The first approach to solve the SBH problem was a branch & bound method proposed in [3]. However, this approach becomes unpractical with growing problem size. For example, the algorithm was only able to solve 1 out of 40 different problem instances concerning DNA target sequences with 200 nucleotide bases within one hour. Another argument against this branch & bound algorithm is the fact that an optimal solution to the SBH problem does not necessarily provide a DNA sequence that is equal to the target sequence. Therefore, the importance of finding *optimal* solutions is not the same as for other optimization problems. Therefore, the research community has focused on heuristic techniques for tackling the SBH

Table 2.1: A list of approaches for the SBH problem.

Type of algorithm	**Identifier**	**Publication**
Constructive heuristic	LAG	Błażewicz et al. [3], 1999
Constructive heuristic	OW	Błażewicz et al. [2], 2002
Evolutionary algorithm	EA1	Błażewicz et al. [8, 6], 2002
Evolutionary algorithm	EA2	Endo [23], 2004
Evolutionary algorithm	EA3	Bui and Youssef [14], 2004
Tabu search	TS	Błażewicz et al. [4], 2000
Tabu search / scatter search hybrid	TS/SS	Błażewicz et al. [5, 6], 2004

problem. Most of the existing approaches are meta-heuristics such as evolutionary algorithms and tabu search techniques. A list of existing approaches for the SBH problem is given in Table 2.1. We also want to remark at this point that some approaches exist for an easier version of the SBH problem in which the first oligonucleotide of the target sequence is given (see for example [24]).

Chapter 3

Constructive Heuristics

Despite the fact that well-working constructive heuristics are often the basis for well-working meta-heuristics, only two constructive heuristics exist for the DNA sequencing problem by hybridization. Both approaches were proposed by Błażewicz and colleagues; the first one is a Look-Ahead Greedy (LAG) that was proposed in [3], and the second one called OW was proposed in [2]. In this chapter we describe new types of constructive heuristics that will be used later for the development of well-working ACO approaches. The contents of this chapter are published in [11].

3.1 The LAG Heuristic

The constructive heuristic that we describe in the following is a version of the look-ahead greedy (LAG) heuristic proposed in [3].

The idea of LAG is to start the path construction in graph G (see Section 2.2.2 for the definition of G) with one of the probes of the spectrum, and to extend this path in a step-by-step manner by means of a look-ahead strategy. The way in which this is done is shown in Algorithm 1. In this algorithm—as well as in the other algorithms outlined in this section—the following notations are used; given a subset $\hat{S} \subseteq S$ of the spectrum S, and a nucleotide $s \in \hat{S}$, we define:

$$\text{bpre}(s) := \text{argmax}\{o_{s',s} \mid s' \in \hat{S},\ s' \neq s\} \ , \tag{3.1}$$

$$\text{bsuc}(s) := \text{argmax}\{o_{s,s'} \mid s' \in \hat{S},\ s' \neq s\} \ , \tag{3.2}$$

In words, $\text{bpre}(s)$ is the best available predecessor for s in $\hat{S}$, that is, the oligonucleotide that—as a predecessor of s—has the biggest overlap with s. Accordingly, $\text{bsuc}(s)$ is the best available successor for s in $\hat{S}$. In case of ties, the first one that is found is taken.

The most important part of this heuristic is the look-ahead strategy that is used to chose which node to add to a path p. This strategy consist in finding the oligonucleotide $s^* \in \hat{S}$ which maximizes the following function:

$$s^* := \text{argmax}\{o_{p[l(p)],s} + o_{s,\text{bsuc}(s)} \mid s \in \hat{S}\} \tag{3.3}$$

In words, the heuristic will add to the end of the path p the oligonucleotide s^* which has a good overlap with the last node in p (i.e., $p[l(p)]$) and which also has a good best successor.

Algorithm 1 Our LAG heuristic

input: A graph $G = (S, E)$, and the length of the DNA target sequence n
$s^* :=$ Choose_Initial_Oligonucleotide(S)
$p := \langle s^* \rangle$
$\hat{S} := S \setminus \{s^*\}$
while $c(p) < n$ **do**
  $s^* := \text{argmax}\{o_{p[l(p)],s} + o_{s,\text{bsuc}(s)} \mid s \in \hat{S}\}$
  Extend path p by adding s^* to its end
  $\hat{S} := \hat{S} \setminus \{s^*\}$
end while
$p :=$Find_Best_Subpath(p)
output: DNA sequence s that is obtained from p

Remember that $l(p)$ is the length of the path p (i.e., the number of nodes in the path), and $c(p)$ denotes the length of the DNA sequence derived from p.

In the original version of LAG as presented in [3], the function Choose_Initial_Oligonucleotide(S) chooses a random vertex for starting the path construction. However, we implemented this function as follows. First, set $S_{bs} \subset S$ is defined as the set of all oligonucleotides in S whose best successor is better or equal to the best successor of all the other oligonucleotides in S.

$$S_{bs} := \{s \in S \mid \forall s' \in S,\ o_{s,\text{bsuc}(s)} \geq o_{s',\text{bsuc}(s')}\} \tag{3.4}$$

Then, set $S_{wp} \subseteq S_{bs}$ is defined as the set of all oligonucleotides in S_{bs} whose best predecessor is worse or equal to the best predecessor of all the other oligonucleotides in S_{bs}:

$$S_{wp} := \{s \in S_{bs} \mid \forall s' \in S_{bs},\ o_{\text{bpre}(s),s} \leq o_{\text{bpre}(s'),s'}\} \tag{3.5}$$

As starting oligonucleotide we choose the one (from S_{wp}) that is found first. The idea hereby is to start the path construction with an oligonucleotide that has a very good successor and at the same time a very bad predecessor. Such an oligonucleotide has a high probability to coincide with the start of the DNA target sequence s_t.

Finally, in case $c(p) > n$, function Find_Best_Subpath settles for the longest (in terms of the number of oligonucleotides) subpath p' of p such that $c(p') \leq n$, and replaces p by p'.

Example

Let us use the example described in Section 2.2.3 to exemplify the LAG method. In this example, the target sequence was s_t = ACTGACTC and the spectrum obtained from the hybridization experiment was S = {ACT,TGA,GAC,CTC,TAA}.

In the first construction step of LAG, the first node is chosen. First S_{bs} is computed; S_{bs} := {ACT,TGA,GAC} because ACT, TGA and GAC are the probes which have a perfect successor. Then S_{wp} is computed; S_{wp} := {TGA}. Therefore the initial part of the sequence is s := TGA and the initial path $p := \langle$TGA$\rangle$.

In the second step, a probe in $\hat{S}$ = {ACT,GAC,CTC,TAA} must be chosen to be added at the end of p. Figure 3.1 shows the overlap—according to the look ahead function 3.3—of the different possibilities. GAC is the one with the biggest overlap, therefore it is added at the end of p (i.e., $p := \langle$TGA, GAC$\rangle$).

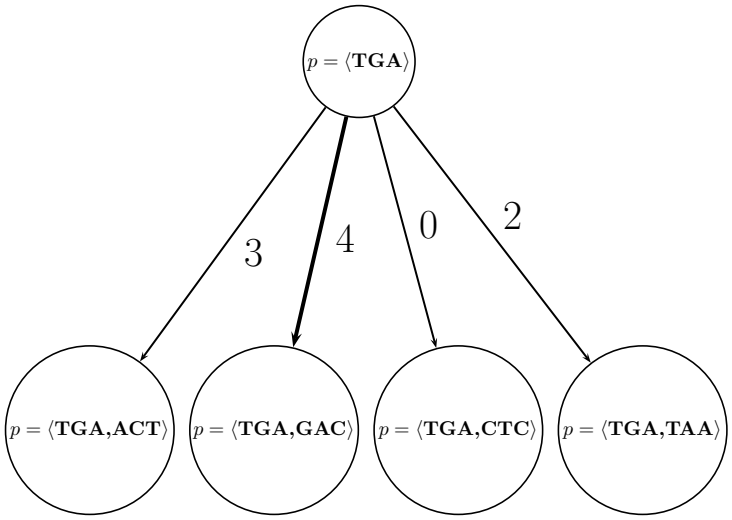


Figure 3.1: Overlap of the different possibilities of path extension in the second step of the LAG example

In further steps of the algorithm ACT and CTC are added to the end of the path (thus, $p := \langle$TGA,GAC,ACT,CTC$\rangle$). Finally TAA is added at the end of path but, afterwards, it is removed from the path in the Find_Best_Subpath(p) method. Furthermore, the solution of LAG is TGACTC. All the construction steps are described graphically in Figure 3.2.

3.2 The SH Heuristic

In order to study if the "look-ahead" mechanism of LAG is really useful, we propose a second heuristic for the SBH problem: the SH heuristic.

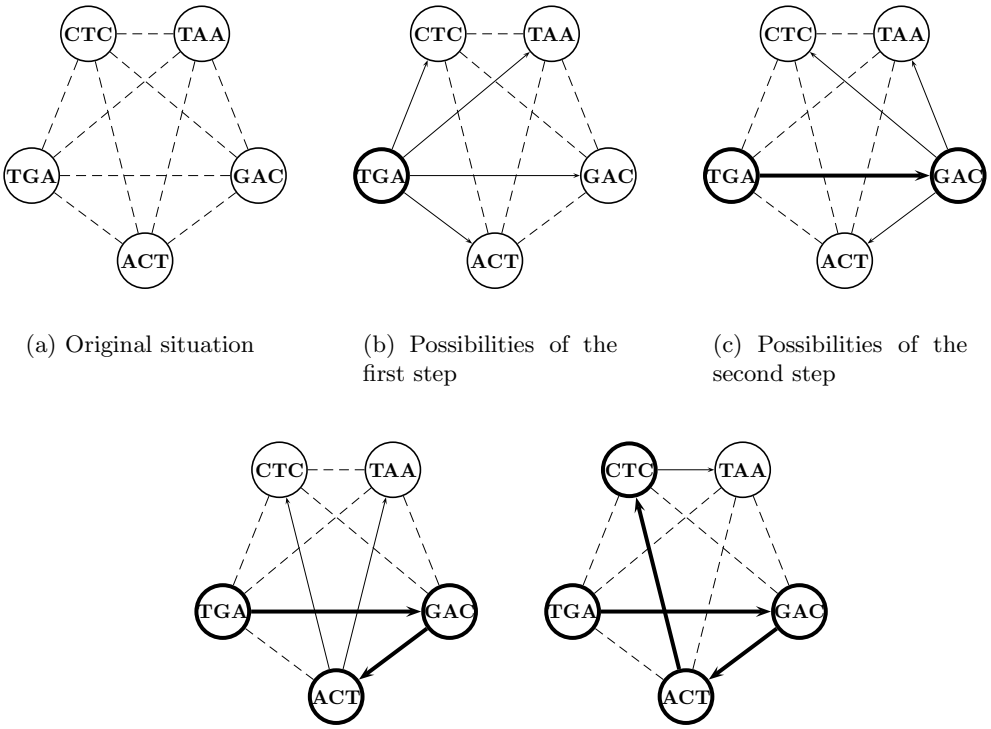
SH is just a simplification of the LAG heuristic where SH stands for Simple Heuristic. This heuristic is identical to LAG except that SH does not use the look-ahead strategy for the choice of the respective next node. Instead, this heuristic adds the node which has the best overlap with the so-far constructed path (i.e., the oligonucleotide $s \in \hat{S}$ which maximizes $o_{p[l(p)],s}$); in other words, the choice is on the best successor of the last probe of the path (i.e., bsuc($p[l(p)]$)). The complete procedure is shown in Algorithm 2, where the procedures Choose_Initial_Oligonucleotide(S) and Find_Best_Subpath(p) are as explained for the LAG heuristic in the previous section.

Example

Let us use the example described in Section 2.2.3, to exemplify also the SH method. In this example, the target sequence was s_t = ACTGACTC and the spectrum obtained from the hybridization experiment was $S = \{$ACT,TGA,GAC,CTC,TAA$\}$.

In the first construction step of SH , the first node is chosen. As in the LAG example, the first node chosen is $s :=$ TGA and the initial path $p := \langle$TGA$\rangle$.

In the second step, a probe in $\hat{S} = \{$ACT,GAC,CTC,TAA$\}$ must be chosen to be added at the end of p. Figure 3.3 shows the overlaps—according to the non look ahead function—of the different possibilities. GAC is the one with the biggest overlap, therefore it is added at the end of p (i.e., $p := \langle$TGA, GAC$\rangle$).



(a) Original situation

(b) Possibilities of the first step

(c) Possibilities of the second step

(d) Possibilities of the third step

(e) Final solution

Figure 3.2: Construction steps of the LAG example. The bold nodes and edges represent, respectively, oligonucleotides and connections in the solution path. Solid but not bold edges represent the possible choices of the corresponding step. Dashed edges are the remaining edges of G

In further steps of the algorithm ACT and CTC are added to the end of the path (thus, $p := \langle$TGA,GAC,ACT,CTC$\rangle$). Finally TAA is added at the end of path but, afterwards, it is removed from the path in the Find_Best_Subpath(p) method. Furthermore, the solution of SH is TGACTC. In this case both LAG and SH have followed the same construction steps to solve the problem, therefore both have obtained the same solution.

3.3 The FB-LAG heuristic

FB-LAG is a simple extension of the LAG heuristic obtained by allowing the path construction not only in forward direction but also in backward direction. We call this heuristic henceforth Forward-Backward Look-Ahead Greedy (FB-LAG) heuristic. At each construction step the heuristic decides (with the same criterion as LAG) to extend the current path either in forward direction (i.e., from the tail of the path) or in backward direction (i.e., from the head of the path).

A second change with respect to LAG concerns the implementation of the function Choose_Initial_Oligonucleotide(S). As the path construction allows forward and backward construction,

Algorithm 2 The SH heuristic

input: A graph $G = (S, E)$, and the length of the target sequence n
$s^* :=$ Choose_Initial_Oligonucleotide(S)
$p := \langle s^* \rangle$
$\hat{S} := S \setminus \{s^*\}$
while $c(p) < n$ **do**
  $s^* :=$ bsuc($p[l(p)]$)
  Extend path p by adding s^* to its end
  $\hat{S} := \hat{S} \setminus \{s^*\}$
end while
$p :=$Find_Best_Subpath(p)
output: DNA sequence s that is obtained from p

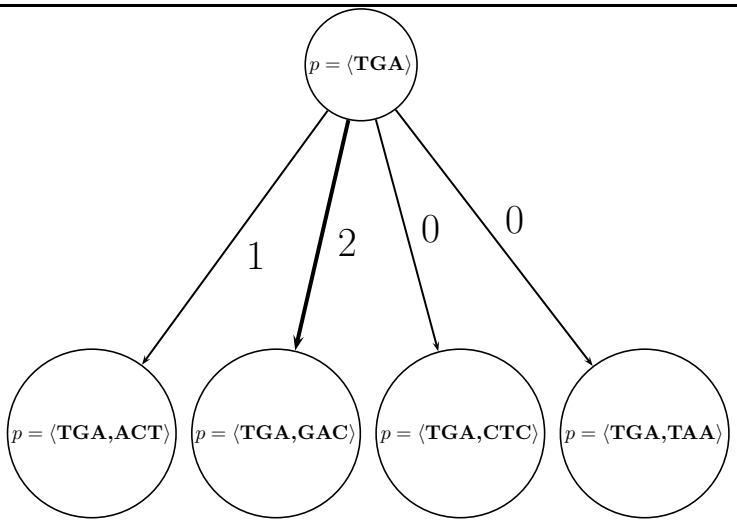


Figure 3.3: Overlap of the different possibilities of path extension in the second step of the SH example

it is not necessary to start the path construction with an oligonucleotide that has a high probability of being the beginning of the DNA target sequence. It is more important to start with an oligonucleotide of the spectrum S that has a high probability of being part of the DNA target sequence.

Thus the initial oligonucleotide s^* is chosen in the following way:

$$s^* := \text{argmax}\{o_{\text{bpre}^2(s),\text{bpre}(s)} + o_{\text{bpre}(s),s} + o_{s,\text{bsuc}(s)} + o_{\text{bsuc}(s),\text{bsuc}^2(s)} \mid s \in S\} \ , \qquad (3.6)$$

where $\text{bpre}^2(s)$ denotes the best predecessor of the best predecessor of s (i.e., $\text{bpre}^2 \equiv \text{bpre}(\text{bpre}(s))$), and similar for $\text{bsuc}^2(s)$ (i.e., $\text{bsuc}^2 \equiv \text{bsuc}(\text{bsuc}(s))$). Algorithm 3 details the procedure of the FB-LAG heuristic.

Example

Let us use the same sequence as in the LAG example; the target sequence was $s_t =$ ACTGACTC and the spectrum was {ACT,TGA,GAC,CTC,TAA}. All the construction steps are described graphically in Figure 3.5.

Algorithm 3 The FB-LAG heuristic

input: A graph $G = (S, E)$, and the length of the target sequence n
$s^* :=$ Choose_Initial_Oligonucleotide(S)
$p := \langle s^* \rangle$
$\hat{S} := S \setminus \{s^*\}$
while $c(p) < n$ **do**
  $s_r := \text{argmax}\{o_{p[l(p)],s} + o_{s,\text{bsuc}(s)} \mid s \in \hat{S}\}$
  $s_l := \text{argmax}\{o_{\text{bpre}(s),s} + o_{s,p[1]} \mid s \in \hat{S}\}$
  if $o_{p[l(p)],s_r} + o_{s_r,\text{bsuc}(s)} > o_{\text{bpre}(s),s_l} + o_{s_l,p[1]}$ **then**
    Extend path p by adding s_r to its end
    $\hat{S} := \hat{S} \setminus \{s_r\}$
  else
    Extend path p by adding s_l to its beginning
    $\hat{S} := \hat{S} \setminus \{s_l\}$
  end if
end while
$p :=$Find_Best_Subpath(p)
output: DNA sequence s that is obtained from p

In the first construction step the first node is chosen. TGA and GAC are the two probes that maximize Equation 3.6. Let us suppose that GAC is the first one found. Therefore, GAC is chosen to be the initial path (i.e., $p := \langle$GAC$\rangle$).

In the second step, a probe in $\hat{S} = \{$ACT,TGA,CTC,TAA$\}$ must be chosen to be added at the beginning or at the end of the path. Figure 3.4 shows the overlap of the different possibilities. Inserting ACT at the end of p is the one with biggest overlap, therefore it is added at the end of $p = \langle$GAC,ACT$\rangle$.

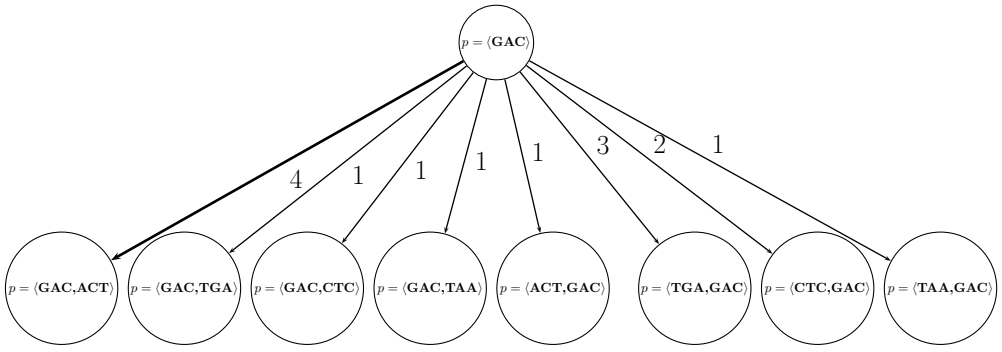
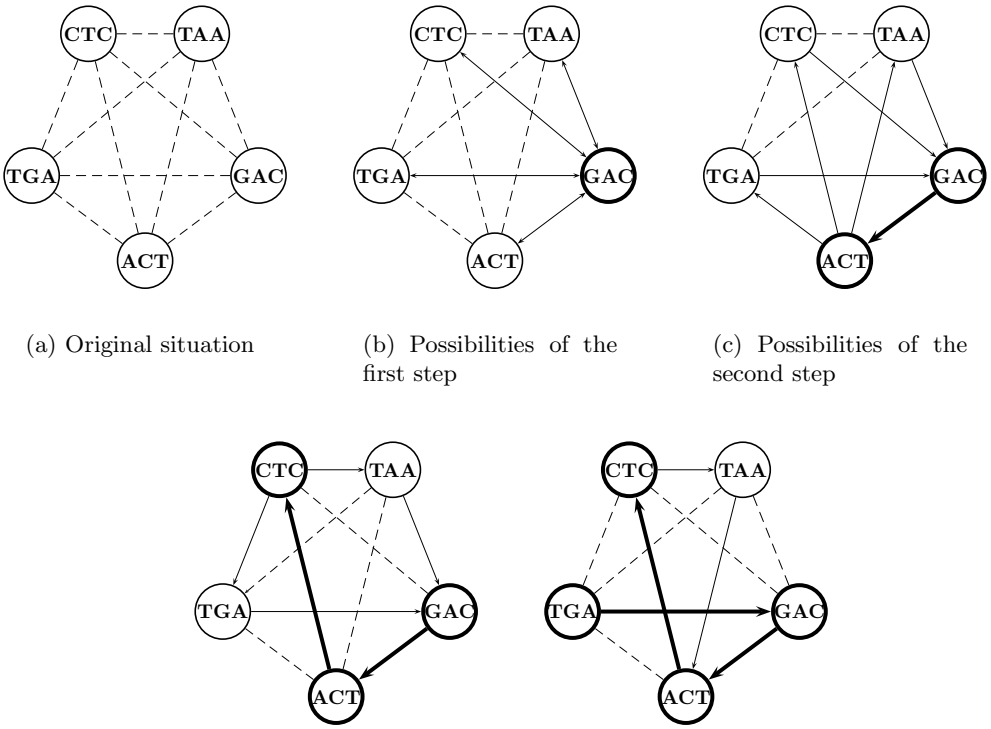


Figure 3.4: Overlap of the different possibilities of path extension in the second step of the FB-LAG example

In further steps CTC is added at the end of the path (i.e., $p := \langle$GAC,ACT,CTC$\rangle$), and TGA is added at the beginning of the resulting path (i.e., $p := \langle$TGA,GAC,ACT,CTC$\rangle$). Therefore the solution sequence of FB-LAG is TGACTC.



(a) Original situation

(b) Possibilities of the first step

(c) Possibilities of the second step

(d) Possibilities of the third step

(e) Final solution

Figure 3.5: Construction steps of the FB-LAG example. The bold nodes and edges represent, respectively, used oligonucleotides and connections in the solution path. Solid but not bold edges represent the possible choices of the step. Dashed edges are the remaining edges of G.

3.4 The FB-SH heuristic

Corresponding to FB-LAG , we also propose the extension of SH. The resulting Forward-Backward Simple Heuristic (FB-SH), is a simplification of the FB-LAG heuristic. It differs from SH by allowing the path construction to both sides. Therefore, in every construction it chooses between the best predecessor of the first node of the path, and the best successor of the last node of the path, in an attempt to maximize the overlap. The complete procedure is shown in Algorithm 4, where the procedure Choose_Initial_Oligonucleotide(S) and Find_Best_Subpath(p) are as explained in the FB-LAG heuristic in the previous section.

Example

Let us use the same sequence as in the FB-LAG example; the target sequence was s_t = ACTGACTC and the spectrum was {ACT,TGA,GAC,CTC,TAA}.

In the first construction step the first node is chosen. As in FB-LAG heuristic, GAC is chosen to be the initial path (i.e., $p := \langle \text{GAC} \rangle$).

In the second step, a probe in $\hat{S}$ = {ACT,TGA,CTC,TAA} must be chosen to be added at the beginning or at the end of the path. Figure 3.6 shows the overlap of the different

Algorithm 4 The FB-SH heuristic

input: A graph G, and the length of the target sequence n
$s^* :=$ Choose_Initial_Oligonucleotide(S)
$p := \langle s^* \rangle$
$\hat{S} := S \setminus \{s^*\}$
while $c(p) < n$ **do**
  $s_r :=$ bsuc($p[l(p)]$)
  $s_l :=$ bpre($p[1]$)
  if $o_{p[l(p)],s_r} > o_{s_l,p[1]}$ **then**
    Extend path p by adding s_r to its end
    $\hat{S} := \hat{S} \setminus \{s_r\}$
  else
    Extend path p by adding s_l to its beginning
    $\hat{S} := \hat{S} \setminus \{s_l\}$
  end if
end while
$p :=$Find_Best_Subpath(p)
output: DNA sequence s that is obtained from p

possibilities. Inserting ACT at the end of p and adding TGA at the beginning of p are the two choices with biggest overlap. Let us assume that ACT is the first found. Therefore it is added at the end of $p = \langle$GAC,ACT$\rangle$.

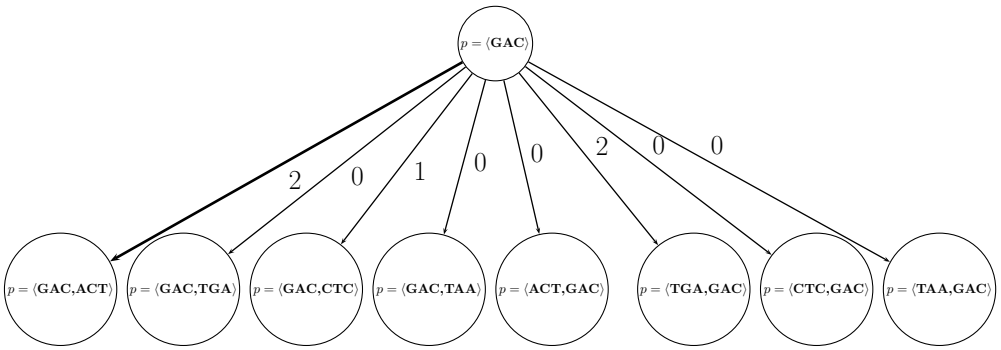


Figure 3.6: Overlap of the different possibilities of path extension in the second step of the FB-SH example

In further steps TGA is added at the beginning of the path (i.e., $p := \langle$TGA,GAC,ACT$\rangle$) and CTC at the end of the resulting path (i.e., $p := \langle$TGA,GAC,ACT,CTC$\rangle$). Therefore the resulting DNA sequence of FB-SH is TGACTC.

3.5 The SM Heuristic

This is the fifth heuristic we propose for the SBH problem. The idea of the sub-sequence merger (SM) heuristic (see Algorithm 5) is conceptionally quite different to the LAG, SH, FB-LAG and FB-SH heuristics. Instead of constructing one single path, the SM heuristic starts with a set P of $|S|$ paths (i.e., $P = \{\langle s \rangle | s \in S\}$), each of which only contains exactly one

oligonucleotide $s \in S$, and then merges paths until a path of sufficient size is obtained. The heuristic works in two phases. In the first phase, two paths p and p' can only be merged if p' is the unique best successor of p, and if p is the unique best predecessor of p'. The heuristic enters into the second phase if and only if, the first phase has not already produced a path of sufficient length. In the second phase, the uniqueness conditions are relaxed, that is, two paths p and p' can be merged if p' is among the best successors of p, and p is among the best predecessors of p'. The reason of having two phases is the following: The first phase aims to produce possibly error free sub-sequences of the DNA target sequence, whereas the second phase (which is more error prone due to the relaxed uniqueness condition) aims at connecting the sub-sequences produced in the first phase in a reasonable way.

In Algorithm 5, given two paths p and p', $o_{p,p'}$ is defined as $o_{p[l(p)],p'[1]}$, that is, $o_{p,p'}$ is the overlap of the last oligonucleotide in p with the first one in p'. In correspondence to the notations introduced in Equations 3.1 and 3.2, the following notations are used:

$$\text{bsuc}(p) := \text{argmax}\{o_{p,p'} \mid p' \in P,\ p' \neq p\}\ , \tag{3.7}$$

$$\text{bpre}(p) := \text{argmax}\{o_{p',p} \mid p' \in P,\ p' \neq p\}\ . \tag{3.8}$$

where P is the set of paths.

Furthermore, $S_{\text{bsuc}}(p)$ is defined as the set of best successor paths of p, that is,

$$S_{\text{bsuc}}(p) := \{p' \in P \mid o_{p,p'} = o_{p,\text{bsuc}(p)}\}, \tag{3.9}$$

and $S_{\text{bpre}}(p)$ is defined as the set of best predecessors of p, that is,

$$S_{\text{bpre}}(p) := \{p' \in P \mid o_{p',p} = o_{\text{bpre}(p),p}\}. \tag{3.10}$$

Finally, function Find_Best_Subpath(p) is implemented as outlined before.

Example

Let us consider, as an example, the target sequence s_t = ACTAGGG. Assuming $l = 3$, the ideal spectrum is {ACT,CTA,TAG,AGG,GGG}. However, let us assume that the hybridization experiment provides us with the following faulty spectrum S = {ACT,CTA,TAG,GGG, CTT, TTT}. This spectrum has one negative error, because AGG does not appear in S. Furthermore, S has two positive errors, because it includes the oligonucleotides CTG and TTT, which do not appear in the target sequence. Let us use this example to show the steps of SM.

In the first phase CTA with TAG and CTT with TTT are merged creating p_0 := ⟨CTA, TAG⟩ and p_1 := ⟨CTT, TTT⟩. The resulting p_0 := ⟨CTA, TAG⟩ is merged with ⟨GGG⟩. In the first phase ⟨ATC⟩ can not be merged neither with ⟨CTA⟩ nor ⟨CTT⟩ because they both have the same overlap.

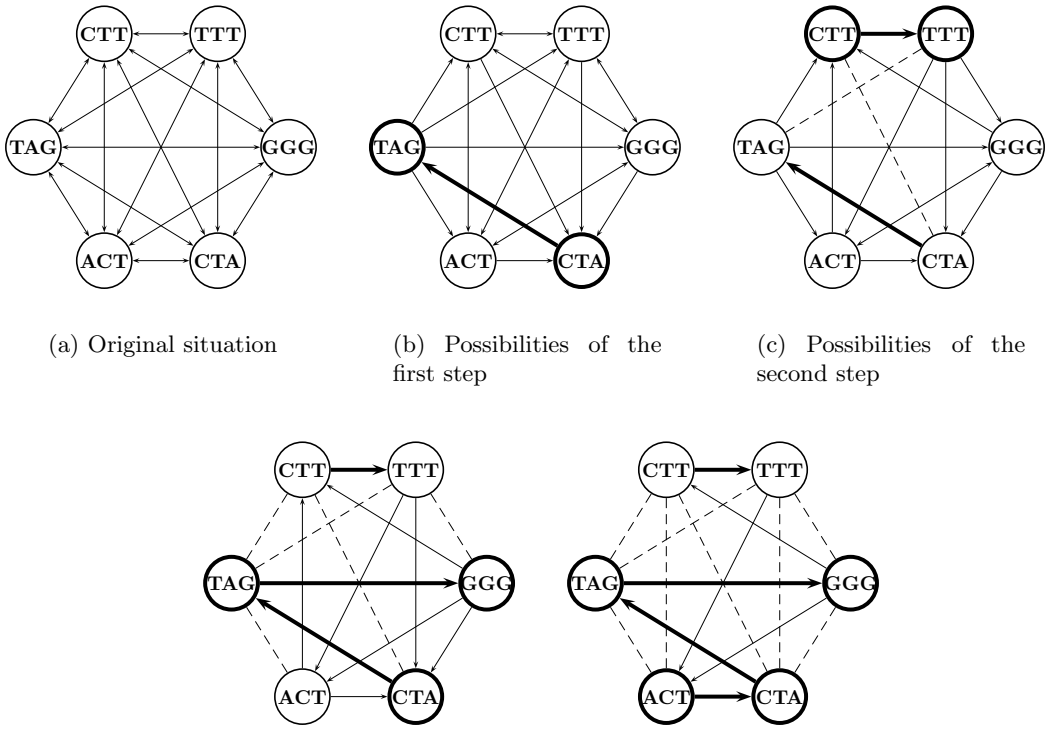
In the second phase two options are presented; merging p_1 = ⟨ACT⟩ with p_2 = ⟨CTA, TAG, GGG⟩ or with p_3 = ⟨CTT,TTT⟩. As the the length of p_1 is greater than the length of p_2, the first one is chosen (i.e., $l(p_1)+l(p_2) > l(p_1)+l(p_3)$). The resulting final path is ⟨ACT, CTA, TAG, GGG⟩ which represents the target sequence.

Algorithm 5 The SM heuristic

input: A graph $G = (S, E)$, and the length of the target sequence n
$P := \{\langle s \rangle \mid s \in S\}$

PHASE 1:
stop := FALSE
for *overlap* $:= l - 1,$ **to** 1 **do**
  while $\exists\, p, p' \in P$ **s.t.** $o_{p,p'} \geq$ *overlap* **&** $|S_{\text{bsuc}}(p)| = 1$ **&** $|S_{\text{bpre}}(p')| = 1$ **&** $\text{bsuc}(p) = p'$ **&** $\text{bpre}(p') = p$ **&** **not** *stop* **do**
    Add path p' to the end of path p
    $P := P \setminus \{p'\}$
    stop$:= c(p) \geq n$
  end while
end for

PHASE 2:
for *overlap* $:= l - 1,$ **to** 1 **do**
  while $\exists\, p, p' \in P$ **s.t.** $o_{p,p'} \geq$ *overlap* **&** $p' \in S_{\text{bsuc}}(p)$ **&** $p \in S_{\text{bpre}}(p')$ **&** **not** *stop* **do**
    Choose p and p' such that $l(p) + l(p')$ is maximal
    Add path p' to the end of path p
    $P := P \setminus \{p'\}$
    stop $:= c(p) \geq n$
  end while
end for
Let p be the path in P with maximal cost
$p :=$ Find_Best_Subpath(p)
output: DNA sequence s that is obtained from p



(a) Original situation

(b) Possibilities of the first step

(c) Possibilities of the second step

(d) Possibilities of the third step

(e) Final solution

Figure 3.7: Construction steps of the SM example. The bold edges represent all the nodes which are merged in a path, and the bold nodes represent the last path that has been constructed. Solid but not bold edges represent the possible choices of the step. Dashed edges are the remaining edges of G.

3.6 The HSM Heuristic

The sixth heuristic we propose for the SBH problem is the Hybrid Sub-sequence Merger (HSM) heuristic, which is obtained by combining the FB-LAG heuristic with the SM heuristic. This combination is based on the following observation: At every stage of the SM heuristic, the FB-LAG heuristic can be applied to the problem instance that is obtained as follows. Given the current path set P of the SM heuristic, a spectrum $\hat{S}$ is created that contains the DNA sequences retrieved from the paths in P.[1] The result of the FB-LAG heuristic when applied to such a problem instance can (of course) be regarded as a solution to the original problem instance. The complete procedure is shown in Algorithm 6.

It remains to specify at which stages of the SM heuristic the FB-LAG heuristic is applied. The first application of FB-LAG is the one to the original problem instance, that is, before the first phase of SM has started. Then, in the first as well as in the second phase of SM, FB-LAG is applied at the end of the respective for-loop (i.e., after line 11 and after line 21 in Algorithm 5). However, FB-LAG is only applied if the while-loop before was executed at

[1]Note that the oligonucleotides of such a spectrum might have (1) length$> l$, and (2) different lengths.

least once. Note that in case the while-loop is not even executed a single time, the problem instance derived from the path set P has not changed since the previous application of FB-LAG. Finally, the output of HSM is the best result among the different applications of FB-LAG and the final result of SM.

3.7 The S-HSM heuristic

The last heuristic we propose is the the Simple Hybrid Sub-sequence Merger (S-HSM) heuristic which is obtained by combining the FB-SH heuristic with the SM heuristic. It is the same heuristic as HSM, except that HSM uses FB-SH instead of FB-LAG (i.e., in lines 3, 16 and 33 of Algorithm 6 the heuristic used is FB-SH instead of FBLAG).

3.8 Experimental results

In this section we report on some experiments that were performed in order to evaluate the quality of the proposed heuristics.

We implemented the 7 heuristics outlined before in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with AMD64X2 4400 processor and 4 Gb of memory.

A wide-spread set of benchmark instances for DNA sequencing by hybridization was introduced by Błażewicz et al. in [3]. It consists of DNA target sequences coding human proteins obtained from GenBank, which is a database of genetic sequences provided by the National Institutes of Health, USA.[2] The instance set consists of 40 DNA target sequences of length 109, 209, 309, 409, and 509 (altogether 200 instances). Based on real hybridization experiments, the spectra were generated with probe size $l = 10$. All spectra contain 20% negative errors as well as 20% positive errors. For example, the spectra concerning the DNA target sequences of length 109 contain 100 oligonucleotides of which 20 oligonucleotides do not appear in the target sequences.

We applied the 7 heuristics outlined in the previous sections to all problem instances. The results are shown in Tables 3.1 to 3.7. Each table contains the results of the corresponding heuristic averaged over the 40 problem instances of each of the five different sizes. The second row of each table contains the average solution quality (i.e., the average number of oligonucleotides in the constructed paths). Remember that the optimization objective in the SBH problem is to maximize this value. The third table row provides the number (out of 40) of solved problem instances, that is, the number of instances for which a path of maximal length could be found.[3] The fourth and fifth table row provide average similarity scores obtained by comparing the computed DNA sequences with the DNA target sequences. The average scores in the fourth table row are obtained from the Needleman-Wunsch algorithm [29], which is an algorithm for global alignment. In contrast, the average scores that are displayed in the fifth table row are obtained by the application of the Smith-Waterman algorithm [32], which is an algorithm for local alignment. Both algorithms were applied with the following parameters: +1 for a match of oligonucleotides, -1 for a mismatch or a gap. Finally, the sixth table row provides the average computation times for solving one instance (in seconds).

[2]The database access keys for all DNA target sequences are provided in [3].

[3]Remember in this context that an optimal solution to the SBH problem does not necessarily correspond to a DNA sequence that is equal to the target sequence.

Algorithm 6 The HSM heuristic

input: A graph $G = (S, E)$, and the length of the target sequence n
$P := \{\langle s \rangle \mid s \in S\}$
$p_{\text{FB-LAG}}$:= result of the application of FB-LAG to a graph G' created from P
$p^* := p_{\text{FB-LAG}}$
PHASE 1:
stop := FALSE
for *overlap* $:= l - 1,$ **to** 1 **do**
  change := FALSE
  while $\exists\, p, p' \in P$ **s.t.** $o_{p,p'} \geq$ *overlap* **&** $|S_{\text{bsuc}}(p)| = 1$ **&** $|S_{\text{bpre}}(p')| = 1$ **&** $\text{bsuc}(p) = p'$ **&** $\text{bpre}(p') = p$ **&** **not** *stop* **do**
    Add path p' to the end of path p
    $P := P \setminus \{p'\}$
    change := TRUE
    stop := $c(p) \geq n$
  end while
  if *change* **then**
    $p_{\text{FB-LAG}}$:= result of the application of FB-LAG to a graph G' created from P
    if $l(p^*) < l(p_{\text{FB-LAG}})$ **then**
      $p^* := p_{\text{FB-LAG}}$
    end if
  end if
end for
PHASE 2:
for *overlap* $:= l - 1,$ **to** 1 **do**
  change := FALSE
  while $\exists\, p, p' \in P$ **s.t.** $o_{p,p'} \geq$ *overlap* **&** $p' \in S_{\text{bsuc}}(p)$ **&** $p \in S_{\text{bpre}}(p')$ **&** **not** *stop* **do**
    Choose p and p' such that $l(p) + l(p')$ is maximal
    Add path p' to the end of path p
    $P := P \setminus \{p'\}$
    change := TRUE
    stop := $c(p) \geq n$
  end while
  if *change* **then**
    $p_{\text{FB-LAG}}$:= result of the application of FB-LAG to a graph G' created from P
    if $l(p^*) < l(p_{\text{FB-LAG}})$ **then**
      $p^* := p_{\text{FB-LAG}}$
    end if
  end if
end for
Let p_{SM} be the path in P with maximal cost
p_{SM} := Find_Best_Subpath(p_{SM})
if $l(p^*) < l(p_{\text{SM}})$ **then**
  $p^* := p_{\text{SM}}$
end if
output: DNA sequence s that is obtained from p^*

From the results that are displayed in Tables 3.1 to 3.7 we can draw the following conclusions. First, and quite surprisingly, LAG and SH get quite similar results. In fact SH gets better results in most of the runs (specially in small instances). Therefore, this indicates that the look-ahead strategy is not effective in this problem. Second, the results of FB-LAG improve in general over the results of LAG (respectively FB-SH improves over SH). This means that it is beneficial to allow the path construction in two directions (forward as well as backward). Third, the results of the SM heuristic are clearly better than the results of the lineal construction heuristics (LAG , FB-LAG , SH and FB-SH), specially in the similarity score and in the number of solved instances. Forth, the best results are obtained by the hybrid heuristics (both HSM and S-HSM). When comparing between them we can see that in this case, the use of the look-ahead strategy is beneficial, because HSM improves over S-HSM . Even for the largest problem instances, the HSM heuristic produces sequences with very high similarity scores.

In order to provide a comparison of all existing constructive heuristics we added the OW heuristic [2] to this comparison. This comparison is shown graphically in Figure 3.8. The results clearly show that HSM is currently the best available constructive heuristic. Finally, in Figure 3.9 we present a comparison between HSM and the best available meta-heuristic approaches from the literature, namely, the evolutionary algorithms EA1 to EA3, the tabu search TS, and the hybrid tabu search with scatter search TS/SS, all of them are cited in section 2.3.2. The results are surprising: HSM is clearly better than the 4 meta-heuristic approaches EA1, EA3, TS, and TS/SS. Furthermore, the results of HSM are—except for the problem instances of target sequence size 509—comparable to the results of the best meta-heuristic approach EA2. Taking into account the advantage in computation time (i.e., HSM needs not even half a second to compute its results for the largest problem instances, while the meta-heuristics need between several seconds and several minutes) the HSM heuristic seems to be a good choice even when compared to meta-heuristic approaches.

Table 3.1: Results of LAG for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	76.98	153.53	230.68	309.03	383.08
Solved instances	23	15	12	7	4
Average similarity score (global)	77.05	133.63	171.78	206.80	218.60
Average similarity score (local)	91.83	152.43	209.33	272.40	293.48
Average computation time (sec)	0.0035	0.016	0.037	0.076	0.13

Table 3.2: Results of SH for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	77.20	152.77	229.00	302.72	375.27
Solved instances	26	18	18	7	5
Average similarity score (global)	80.85	142.10	199.97	214.05	269.67
Average similarity score (local)	95.27	167.20	234.80	277.32	334.90
Average computation time (sec)	0.0027	0.012	0.024	0.042	0.068

Table 3.3: Results of FB-LAG for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	78.38	155.70	234.95	310.03	386.20
Solved instances	32	17	18	7	1
Average similarity score (global)	99.78	153.03	225.45	241.00	221.83
Average similarity score (local)	102.38	174.15	253.63	284.58	290.13
Average computation time (sec)	0.0051	0.022	0.054	0.11	0.19

Table 3.4: Results of FB-SH for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	77.65	148.27	218.32	299.30	355.65
Solved instances	36	21	19	13	5
Average similarity score (global)	102.27	159.47	213.50	247.20	226.02
Average similarity score (local)	104.12	179.47	252.10	303.67	319.40
Average computation time (sec)	0.004	0.014	0.031	0.059	0.092

Table 3.5: Results of SM for the instances by Błażewicz et al. [3].

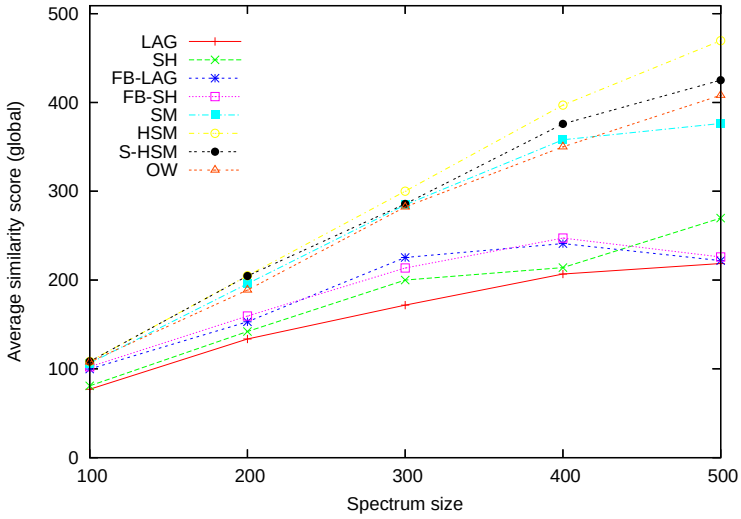
Spectrum size	100	200	300	400	500
Average solution quality	79.75	157.80	234.90	306.90	367.38
Solved instances	38	31	30	28	18
Average similarity score (global)	106.33	195.85	284.68	357.98	376.25
Average similarity score (local)	107.20	203.03	293.75	377.00	416.68
Average computation time (sec)	0.005	0.02	0.046	0.082	0.13

Table 3.6: Results of HSM for the instances by Błażewicz et al. [3].

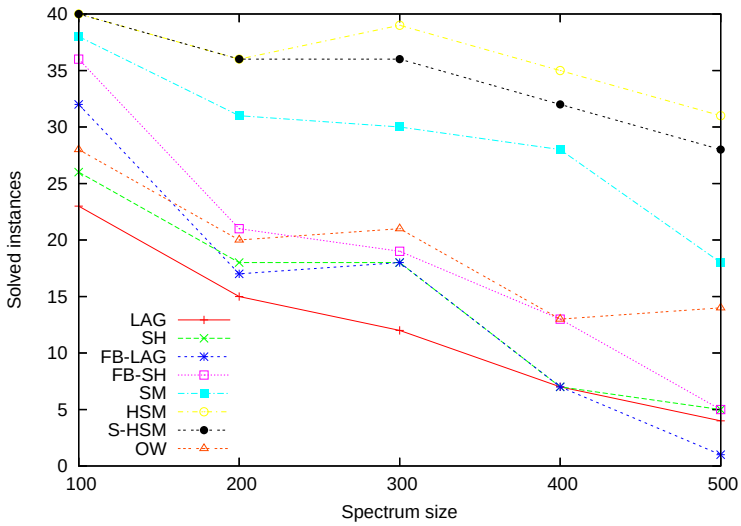
Spectrum size	100	200	300	400	500
Average solution quality	80.00	159.68	239.90	319.38	398.88
Solved instances	40	36	39	35	31
Average similarity score (global)	108.40	204.78	300.00	396.90	469.55
Average similarity score (local)	108.70	206.85	305.35	399.85	479.88
Average computation time (sec)	0.012	0.048	0.11	0.21	0.35

Table 3.7: Results of S-HSM for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	80.00	159.55	238.67	318.40	394.55
Solved instances	40	36	36	32	28
Average similarity score (global)	108.40	204.55	285.62	375.87	425.02
Average similarity score (local)	108.70	206.85	298.72	387.33	456.58
Average computation time (sec)	0.0072	0.032	0.076	0.13	0.23



(a) Global average similarity score



(b) Number of optimally solved instances.

Figure 3.8: Comparison of all existing constructive heuristics. The comparison concerns the instances of Błażewicz et al. [3].

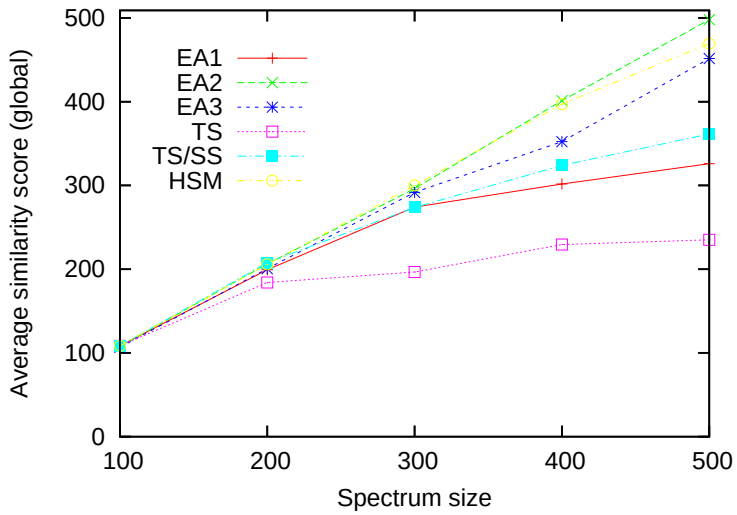


Figure 3.9: Comparison of HSM with meta-heuristics from the literature. The comparison is done concerning the global average similarity score obtained for the instances by Błażewicz et al. [3].

Chapter 4

Ant Colony Optimization: ACO

In section 2.3 we have introduced meta-heuristics as tools for solving Combinatorial Optimization problems. We also mentioned briefly a meta-heuristic named ant colony optimization (ACO) [21] which is the one we will use in this thesis for tackling the SBH problem. In this chapter we will give a more detailed description of ACO.

4.1 Introduction

Ant colonies, and more generally social insect societies, are distributed systems that, in spite of the simplicity of their individuals, present a highly structured social organization. As a result of this organization, ant colonies can accomplish complex tasks that in some cases far exceed the individual capabilities of a single ant.

The field of "ant algorithms" studies models derived from the observation of real ants' behavior, and uses these models as a source of inspiration for the design of novel algorithms for the solution of optimization and distributed control problems.

One of the most successful examples of ant algorithms is known as "ant colony optimization", or ACO. ACO is inspired by the foraging behavior of ant colonies, and targets discrete optimization problems.

4.1.1 Path search: double bridge experiments

The visual perceptive faculty of many ants species is only rudimentary developed and there are ant species that are completely blind. In fact, most of the communication among individuals, or between individuals and the environment, is based on the use of chemicals produced by ants. These chemicals are called *pheromones*. Particularly important for the social life of some ant species is the *trail pheromone*. Trail pheromone is a specific type of pheromone that some ants species use for marking paths on the ground, for example, paths from food sources to the nest. By sensing pheromone trails ants can follow the path to food discovered by other ants.

The foraging behavior of many ant species, as for example, *I. humilis* [16], *Linepithema humile*, and *Lasius niger* [12] is based on indirect communication mediated by pheromones. While walking from food sources to the nest and vice versa, ants deposit pheromones on the ground, forming in this way a pheromone trail. Ants can smell the pheromone and they tend to choose, probabilistically, paths marked by strong pheromone concentrations.

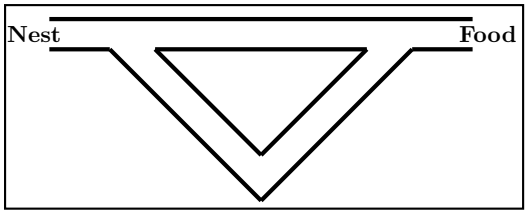
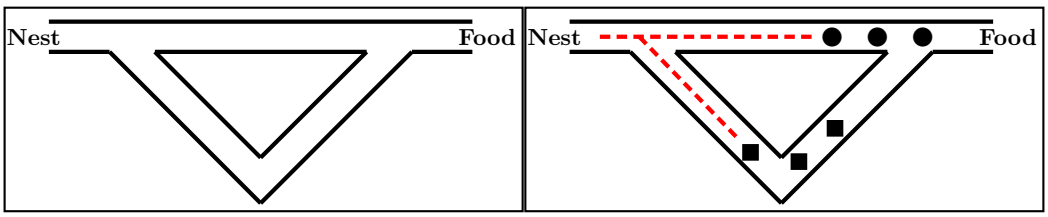


Figure 4.1: Experimental setup for a double bridge experiment where the ratio $r = l_l/l_s$ is greater than 1 (i.e., $l_l > l_s$).

Deneubourg and colleagues [16] did an experiment to study how ants use pheromone trails to create shortest paths form the nest to the food sources. This experiment consisted of a double bridge connecting a nest of ants and a food source. They ran experiments varying the ratio $r = l_l/l_s$ between l_l the length of the longer branch, and l_s the length of the shorter one of the double bridge. Figure 4.1 shows a graphical example of the experimental setup of the experiment.

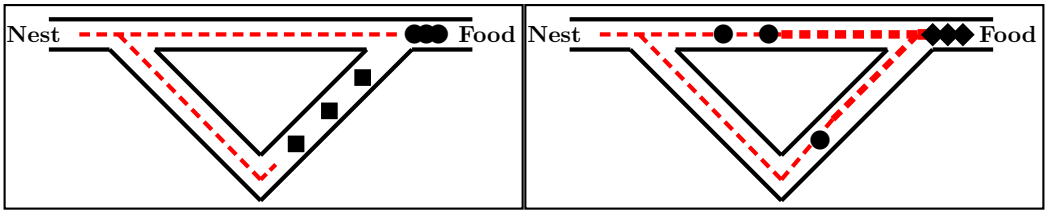
In the first experiment the bridge had two branches of equal length ($r = 1$). At the start, ants were left free to move between the nest and the food source and the percentage of ants that chose one or the other of the two branches were observed over time. The outcome was that although in the initial phase random choices occurred, eventually all the ants used the same branch. This result can be explained as follows. When a trial starts there is no pheromone on the two branches. Hence, the ants do not have a preference and they select with the same probability any of the branches. Yet, because of random fluctuations, a few more ants will select one branch over the other. Because ants deposit pheromone while walking, a larger number of ants on a branch results in a larger amount of pheromone on that branch; this larger amount of pheromone in turn stimulates more ants to choose that branch again, and so on until finally the ants converge to one single path. This auto-catalytic or positive feedback process is, in fact, an example of a self-organizing behavior of the ants: a macroscopic pattern (corresponding to the convergence towards one branch) emerges out of processes and interactions taking place at a "microscopic" level. Another important factor for this behavior is the *pheromone evaporation*. Pheromone deposited evaporates in a certain time. This enables to "erase" pheromone paths.

In the second experiment, the length ratio between the two branches was set to $r = 2$, so that the long branch was twice as long as the short one. In this case, in most of the trials, after some time all the ants chose to use only the short branch. As in the first experiment, ants leave the nest to explore the environment and arrive at a decision point where they have to explore the environment and arrive at a decision point where they have to choose one of the two branches. Because the two branches initially appear identical to the ants, they choose randomly. Therefore, it can be expected that, on average, half of the ants choose the short branch and the other half the long branch. However, this experimental setup presents a remarkable difference with respect to the previous one: because one branch is shorter than the other, the ants choosing the short branch are the first to reach the food and to start their return to the nest. But then, when they must make a decision between the short and the long branch, the higher level of pheromone on the short branch will bias their decision in its favor. Therefore, pheromone starts to accumulate faster on the short branch, which will eventually be used by all the ants because of the auto-catalytic process described previously. Figure 4.2 represents this experiment.



(a) All ants are in the nest. There is no pheromone in the environment.

(b) The foraging starts. In probability, 50% of the ants take the short path (symbolized by circles), and 50% take the long path to the food source (symbolized by diamonds).



(c) The ants that have taken the short path have arrived earlier at the food source. Therefore, when returning, the probability to take again the short path is higher.

(d) The pheromone trail on the short path receives, in probability, a stronger reinforcement, and the probability to take this path grows. Finally, due to the evaporation of the pheromone on the long path, the whole colony will, in probability, use the short path.

Figure 4.2: An experimental setting that demonstrates the shortest path finding capability of ant colonies. Between the ants' nest and the only food source exist two paths of different lengths. In the four graphics, the pheromone trails are shown as dashed lines whose thickness indicates the trails' strength.

4.2 Ant System: an initial algorithm

Inspired by the foraging behavior of real ants and its influence in the emergence of shortest paths in the double-bridge experiments, researchers tried to capture the behavior of real ants in an algorithmic framework. The aim was to be able to apply such a technique to solve discrete optimization problems other than the shortest path problem.

4.2.1 The discretized model

As a first step towards an algorithm for discrete optimization, we present in the following a discretized and simplified model for the double-bridge experiment with real ants described at the previous section. The model consists of a graph $G = (V, E)$, where V consists of two nodes, namely v_s (representing the nest of the ants), and v_d (representing the food source). Furthermore, E consists of two links, namely e_1 and e_2, between v_s and v_d. To e_1 we assign a length of l_1 and to e_2 a length of l_2 such that $l_2 > l_1$. In other words, e_1 represents the short path between v_s and v_d, and e_2 represents the long path. Real ants deposit pheromone on the paths on which they move. Thus, the chemical pheromone trails are modeled as artificial

pheromone trails on the corresponding path e_1 and e_2: path e_1 has pheromone τ_1 and path e_2 has pheromone τ_2. Finally we introduce n_a artificial ants. Each ant behaves as follows:

Starting from v_s (i.e., the nest), an ant chooses with probability

$$p[\text{choosing path } e_i] = \frac{\tau_i}{\tau_1 + \tau_2},\ i \in \{1, 2\}, \tag{4.1}$$

between path e_1 and path e_2 for reaching the food source v_d. Obviously, if $\tau_1 > \tau_2$, the probability of choosing e_1 is higher, and vice-versa. For returning from v_d to v_s, an ant uses the same path as it chose to reach v_d, and it changes the artificial pheromone value associated to the used edge. More in detail, having chosen edge e_i an ant changes the artificial pheromone value τ_i as follows:

$$\tau_i := \tau_i + \frac{Q}{l_i} \tag{4.2}$$

where the positive constant Q is a parameter of the model. In other words the artificial pheromone that is added depends on the length of the chosen path: the shorter the path, the higher the amount of added pheromone.

The foraging of an ant colony is in this model simulated as follows: At each step (or iteration) all the artificial ants are initially placed in node v_s. Then, each ant moves from v_s to v_d as outlined above. Finally the pheromone evaporation in the artificial model is simulated by updating the pheromone of each path $e_i \in \{e_1, e_2\}$ in the following way:

$$\tau_i := (1 - \rho) \cdot \tau_i, \quad i \in \{1, 2\} \tag{4.3}$$

The parameter $\rho \in (0, 1]$ regulates the pheromone evaporation. Finally, all ants conduct their return trip and reinforce their chosen path as outlined above. Except from small differences this model represents the behavior of real ants.

4.2.2 Ant System for the TSP: The first ACO algorithm

The model used in the previous section to simulate the foraging behavior of real ants in the double bridge experiment cannot directly be applied to CO problems. Among other reasons, this is because we associated pheromone values directly to solutions to the problem (i.e., one parameter τ_1 for short the path, and one parameter τ_2 for the long path). This way of modeling implies that the solutions to the considered problem are already known. However, in combinatorial optimization we intend to *find* an unknown optimal solution. Thus, when CO problems are considered, pheromone values are associated to solution components instead. Solution components are expected to be finite and of moderate size. As an example we present the first ant colony-based algorithm, called Ant System (AS)[17, 20], applied to the well known CO problem TSP (Travelling Salesman Problem).

In the TSP is given a completely connected, undirected graph $G = (V, E)$ with edge-weights. The nodes V of this graph represent cities, and the edge weights represent the distances between the cities. The goal is to find a cycle in G that contains each node exactly once (henceforth called tour) and whose length is minimal (i.e., Hamiltonian circuit of minimal length). Thus, the search space $\mathcal{S}$ consists of all tours in G. The objective function value $f(s)$ of a tour $s \in \mathcal{S}$ is defined as the sum of the edge-weights of the edges that are in s.

Concerning the AS approach, the edges of the given TSP graph can be considered solution components and thus for each $e_{i,j}$ is introduced a pheromone value $\tau_{i,j}$. The task of each ant

consists in the construction of a feasible TSP solution (i.e., a feasible tour). In other words, the notion of task of an ant changes from "choosing a path from the nest to the food source" to "constructing a feasible solution to the tackled optimization problem". Note that with this change of task, the notions of nest and food source lose their meaning.

Each ant constructs a solution as follows. First, one of the nodes of the fully-connected input graph is randomly chosen as start node. Then, the ant builds a tour in the TSP graph by moving in each construction step from its current node (i.e., the city in which *she* is located) to another node which *she* has not visited yet. At each step, the traversed edge is added to the solution under construction. When no unvisited nodes are left, the ant closes the tour by moving from *her* current node to the node in which *she* started the solution construction. This way of constructing a solution is performed probabilistically as follows. Assuming the ant to be in node v_i, the subsequent construction step is done with probability

$$p(e_{i,j}) = \frac{\tau_{i,j}}{\sum_{\{k \in \{1,\ldots,|V|\} | v_k \notin T\}} \tau_{i,k}}, \quad \forall j \in \{1,\ldots,|V|\}, v_j \notin T. \tag{4.4}$$

where T is the set of nodes the ant has already visited. For an example of such a solution construction, see Fig 4.3

Once all ants of the colony have completed the construction of their solution, pheromone evaporation is performed as follows:

$$\tau_{i,j} := (1-\rho) \cdot \tau_{i,j}, \ \ \forall \tau_{i,j} \in \mathcal{T} \tag{4.5}$$

where $\mathcal{T}$ is the set of all pheromone values. Then the ants perform their return trip. Hereby, an ant—having constructed a solution s—performs for each $e_{i,j} \in s$ the following pheromone deposit:

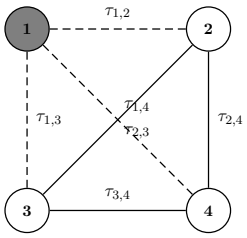
$$\tau_{i,j} := \tau_{i,j} + \frac{Q}{f(s)} \tag{4.6}$$

where Q is again a positive constant and $f(s)$ is the objective function value for the solution s. As explained in the previous section, the system is iterated—applying n_a ants per iteration—until a stopping condition (e.g., a time limit) is satisfied. Even though the AS algorithm has achieved to transform the ants foraging behavior into an algorithm for discrete optimization its performance was not often as good as desired. Therefore, over years, several extensions and improvements of the original AS algorithm were introduced. They are all covered by the definition of the ACO meta-heuristic, which we will outline in the following section.

4.3 The ant colony optimization meta-heuristic

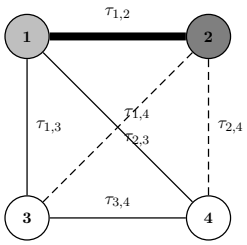
The ACO meta-heuristic, as we know it today, was first formalized by Dorigo and colleagues in 1999 [18]. The definition of the ACO meta-heuristic covers most—if not all—existing ACO variants for discrete optimization problems. In the following, we give a general description of the framework of this more complete ant colony-based meta-heuristic.

The ACO meta-heuristic shares some basic ideas with the initial AS meta-heuristic. Given a CO problem to be solved, one first has to derive a finite set C of solution components which are used to assemble solutions to the CO problem. Second, one has to define a set of *pheromone values* $\mathcal{T}$. This set of values is commonly called the *pheromone model*, which is—seen from



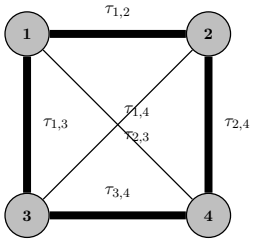
$$p(e_{1,j}) = \frac{\tau_{1,j}}{\tau_{1,2} + \tau_{1,3} + \tau_{1,4}}$$

(a) First step of the solution construction.



$$p(e_{2,j}) = \frac{\tau_{2,j}}{\tau_{2,3} + \tau_{2,3}}$$

(b) Second step of the solution construction.



(c) The complete solution after the final construction step.

Figure 4.3: Example of the solution construction for a TSP problem consisting of 4 cities (modelled by a graph with 4 nodes). The solution construction starts by randomly choosing a start node for the ant; in this case node 1. Figures (a) and (b) show the choices of the first, respectively the second, construction step. Note that in both cases the current node (i.e., location) of the ant is marked by dark gray color, and the already visited nodes are marked by light gray color. The choices of the ants (i.e., the edges *she* may traverse) are marked by dashed lines. The probabilities for the different choices (according to equation 4.4) are given at the right of the graphics. Note that after the second construction step, in which we exemplary assume the ant to have selected node 4, the ant can only move to node 3, and then back to node 1 in order to close the tour.

Algorithm 7 Ant colony optimization (ACO)

```
1: while termination conditions not met do
2:    AntBasedSolutionConstruction()      {see Algorithm 8}
3:    PheromoneUpdate()
4:    DaemonActions()      {optional}
5: end while
```

Algorithm 8 Procedure AntBasedSolutionContruction() of Algorithm 7

```
1: s=⟨⟩
2: Determine 𝒩(s)
3: while 𝒩(s) ≠ ∅ do
4:    c:=ChooseFrom(𝒩(s))
5:    s:=extend s by appending solution component c
6:    Determine 𝒩(s)
7: end while
```

a technical point of view–a parametrized probabilistic model. The pheromone model is one of the central components of the ACO meta-heuristic. The pheromone values $\tau_i \in \mathcal{T}$ are usually associated to solution components or to associations between solution components. The pheromone model is used to probabilistically to generate solutions to the problem under consideration, by assembling them from the set of solution components. In general, similar to AS, the ACO approach attempts to solve an optimization problem by iterating the following two steps:

1. candidate solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space;

2. the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

The pheromone update aims to concentrate the search in regions of the search space containing high quality solutions. In particular, the reinforcement of solution components depending on the solution quality is an important ingredient of ACO algorithms. It implicitly assumes that good solutions consist of good solution components and thus, to learn which components contribute to good solutions can help assembling them into better solutions. In the following, we give a more technical description of the general ACO meta-heuristic whose framework is shown in Algorithm 7.

ACO is an iterative algorithm whose run time is controlled by a principal while-loop as shown in Algorithm 7. In each iteration the three algorithmic components AntBasedSolution-Contruction(), PheromoneUpdate(), and DaemonActions() must be scheduled. In the following we outline these three algorithmic components in detail:

AntBasedSolutionConstruction()

Artificial ants can be regarded as probabilistic constructive heuristics that assemble solutions as sequences of solution components. The finite set of solution components $C = \{c_1, \ldots, c_n\}$

is hereby derived from the discrete optimization problem under consideration.[1] Each solution construction starts with an empty sequence $s = \langle\rangle$ (i.e., with an empty solution). Then, the current sequence s is extended at each construction step by adding a feasible solution component from the set $\mathcal{N}(s) \subseteq C \setminus s$.[2] The specification of $\mathcal{N}(s)$ depends on the solution construction mechanism.[3] The choice of a solution component form $\mathcal{N}(s)$ (see function ChooseFrom($\mathcal{N}(s)$) in algorithm 8) is at each construction step performed probabilistically with respect to the pheromone model. In most ACO algorithms the respective probabilities—also called the *transition probabilities*—are defined as follows:

$$p(c_i \mid s) = \frac{[\tau_i]^\alpha \cdot [\eta(c_i)]^\beta}{\sum_{c_j \in \mathcal{N}(s)} [\tau_j]^\alpha \cdot [\eta(c_j)]^\beta}, \quad \forall c_i \in \mathcal{N}(s), \tag{4.7}$$

where η is an optional weighting function, that is, a function that, sometimes depending on the current sequence, assigns at each construction step a heuristic value $\eta(c_j)$ to each feasible solution component $c_j \in \mathcal{N}(s)$. The values that are given by the weighting function are commonly called the *heuristic information*. Furthermore, the exponents α and β are positive parameters whose values determine the relation between pheromone information and heuristic information. It is interesting to note that when function ChooseFrom($\mathcal{N}(s)$) in Algorithm 8 is implemented to choose deterministically the solution component that maximizes equation 4.7 (i.e., $c := \text{argmax}\{\eta(c_i) | c_i \in \mathcal{N}(s)\}$), we obtain a deterministic greedy algorithm.

This method is detailed in Algorithm 8.

PheromoneUpdate()

Different ACO variants mainly differ in the update of the pheromone values they apply. In the following, we outline a general pheromone update rule in order to provide the basic idea. This pheromone update rule 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 implements a useful form of forgetting pheromone paths, thus favoring the exploration of new areas in the search space. Second, one or more solutions from the current and/or from earlier iterations are used to increase the values of pheromone trail parameters on solution components that are part of these solutions:

$$\tau_i := (1 - \rho) \cdot \tau_i + \rho \cdot \sum_{\{s \in \mathcal{S}_{upd} | c_i \in s\}} w_s \cdot F(s), \tag{4.8}$$

for $i = 1, \ldots, n$. Hereby, $\mathcal{S}_{upd}$ denotes the set of solutions that are used for the update. Furthermore, $\rho \in (0, 1]$ is a parameter called evaporation rate, and $F : \mathcal{S} \mapsto \mathbb{R}^+$ is a so-called quality function such that

$$f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in \mathcal{S}.$$

In other words, if the objective function value of a solution s is better than the objective function value of a solution s', the quality of solution s will be at least as high as the quality

[1] For example, in the case of AS applied to the TSP problem (see previous section), each edge of the fully connected input graph was considered a solution component.

[2] Note that for this set-operation the sequence s is regarded as an ordered set.

[3] In the example of TSP (see previous section) the solution construction mechanism restricted the set of traversable edges to the ones that connected the ants' current node to unvisited nodes.

of solution s'. Equation 4.8 is general enough to allow also an additional weighting of the quality function, since $w_s \in \mathbb{R}^+$ denotes a certain weight of a solution s.

Different versions of this update rule are obtained by different specifications of $\mathcal{S}_{upd}$ and by different weight settings. In many cases, $\mathcal{S}_{upd}$ is composed of some of the solutions generated in the respective iteration (henceforth denoted by $\mathcal{S}_{iter}$), and by the best solution found since the start of the algorithm (henceforth denoted by s_{bs}). Solution s_{bs} is often called the best-so-far solution. A well-known example is the *AS-update* rule, that is, the update rule of AS that we explained in Section 4.2.2. The AS-update rule, which is well-known due to the fact that AS was the first ACO algorithm to be proposed in the literature, is obtained from update rule (4.8) by setting

$$\mathcal{S}_{upd} := \mathcal{S}_{iter} \quad \text{and} \quad w_s = 1, \quad \forall s \in \mathcal{S}_{upd},$$

that is, by using all the solutions that were generated in the respective iteration for the pheromone update, and by setting the weight of each of these solutions to 1. An example of a pheromone update rule that is more used in practice is the *IB-update* rule (where IB stands for *iteration-best*). The IB-update rule is given by

$$\mathcal{S}_{upd} := \{s_{ib} = \text{argmax}\{F(s)|s \in \mathcal{S}_{iter}\}\} \quad \text{with } w_{s_{ib}} = 1,$$

that is, by choosing only the best solution generated in the respective iteration for updating the pheromone values. This solution, denoted by s_{ib}, is weighted by 1. The IB-update rule introduces a much stronger bias towards the good solutions found than the AS-update rule. However, this increases the danger of premature convergence. An even stronger bias is introduced by the *BS-update* rule, where BS refers to the use of the best-so-far solution s_{bs}. In this case $\mathcal{S}_{upd}$ is set to $\{s_{bs}\}$ and s_{bs} is weighted by 1. In practice, ACO algorithms that use variations of the IB-update or the BS-update rule and that additionally include mechanisms to avoid premature convergence, achieve better results than algorithms that use the AS-update rule. Examples are given in the following section.

DaemonActions()

Daemon actions can be used to implement centralized actions which cannot be performed by single ants. Examples are the application of local search methods to the constructed solutions, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon may decide to deposit extra pheromone on the solution components that belong to the best solution found so far.

4.3.1 ACO variants

Even though the original AS algorithm achieved encouraging results for the TSP problem, it was found to be inferior to state-of-art algorithms for the TSP as well as for other CO problems. Therefore, several extensions and improvements of the original AS algorithm were introduced over the years. In the following we outline some of the ACO variants with better results.

$\mathcal{MAX}$-$\mathcal{MIN}$Ant System ($\mathcal{MM}$AS)

One of the most successful ACO variants today is $\mathcal{MAX}$-$\mathcal{MIN}$Ant System ($\mathcal{MM}$AS) [33], which is characterized as follows. $\mathcal{MM}$AS algorithms use an explicit lower bound $\tau_{\min} > 0$

for the pheromone values. In addition to this lower bound, $\mathcal{MM}$AS algorithms use $\tau_{\max} = F(s_{bs})/\rho$ as an upper bound to the pheromone values. The value of this bound is updated each time a new best-so-far solution s_{bs} is found by the algorithm. $\mathcal{MM}$AS algorithms also define a convergence measure. In this work we will use

$$cf := 2 \cdot \left(\left(\frac{\sum_{\tau_{i,j} \in \mathcal{T}} \max\{\tau_{\max} - \tau_{i,j}, \tau_{i,j} - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \tag{4.9}$$

as the convergence factor. This factor is small at the beginning of the algorithm (i.e.,when pheromone values are between $\tau_{\max}$ and $\tau_{\min}$) and increases during the execution of the algorithm (i.e., when pheromone values are similar to $\tau_{\max}$ or $\tau_{\min}$). Depending on this convergence measure, at each iteration either the IB-update or the BS-update rule (both as explained in the previous section) are used for updating the pheromone values. When the convergence factor is small (at the beginning of the algorithm) the IB-update rule is used more often, while during the run of the algorithm, due to an increase of cf, the frequency with which the BS-update rule is used increases.

Ant Colony System (ACS)

Ant Colony System, which was introduced in [19], differs from the original AS algorithm in more aspects than just in the pheromone update. First, instead of choosing at each step during a solution construction the next solution component according to equation 4.7, an artificial ant chooses, with probability q_0, the solution component that maximizes $[\tau_i]^\alpha \cdot [\eta(c_i)]^\beta$, or it performs, with probability $1 - q_0$, a probabilistic construction step, according to equation 4.7. This type of solution construction is called *pseudo-random proportional*. Second, ACS uses the BS-update rule. Third, after each solution construction step, the following additional pheromone update is applied to the pheromone value τ_i whose corresponding solution component c_i was added to the solution under construction:

$$\tau_i := (1 - \xi) \cdot \tau_i + \xi \cdot \tau_0, \tag{4.10}$$

where τ_0 is a small positive constant such that $F_{\min} \geq \tau_0 \geq c$, $F_{\min} := \min\{F(s)|s \in \mathcal{S}\}$, and c is the initial value of the pheromone values. In practice, the effect of this local pheromone update is to decrease the pheromone values on the visited solution components, making in this way these components less desirable for the following ants. This mechanism increases the exploration of the search space within each iteration.

The Hyper-Cube Framework

One of the most recent developments is the Hyper-Cube Framework (HCF) for ACO algorithms [9]. Rather than being an ACO variant, the HCF is a framework for implementing ACO algorithms which is characterized by a pheromone update that is obtained from update rule (4.8) by defining the weight w_s of each solution in $\mathcal{S}_{upd}$ to be

$$w_s = \left(\sum_{\{s \in \mathcal{S}_{upd}\}} F(s) \right)^{-1}$$

Therefore, the pheromone update rule in the HCF is:

$$\tau_i := (1-\rho)\cdot\tau_i + \rho\cdot\sum_{\{s\in\mathcal{S}_{upd}|c_i\in s\}}\frac{F(s)}{(\sum_{\{s'\in\mathcal{S}_{upd}\}}F(s'))} \tag{4.11}$$

This means that ACO variants such as ACS or $\mathcal{MM}$AS can be implemented in the HCF. The HCF comes with several benefits. On the practical side, this framework automatically handles the scaling of the objective function values and limits the pheromone values to the interval $[0,1]$.[4]

The candidate list strategy

In addition to the ACO variants outlined above, the ACO community has developed additional algorithmic features for improving the search process performed by ACO algorithms. A prominent example are so-called candidate list strategies. A candidate list strategy is a mechanism to restrict the number of available choices at each solution construction step. Usually, this restriction applies to a number of the best choices with respect to their transition probabilities (see equation 4.7).

4.3.2 Applying ACO in a multilevel framework (ML-ACO)

Recently, the application of ACO within a general problem solving framework known as the multilevel framework was introduced. Optimization techniques that are based on this framework, i.e., multilevel techniques, have been in use since quite a long time, especially in the area of multigrid methods (see [13] for an overview). More recently, they have been brought into focus by Walshaw for the application to CO. Walshaw and co-workers applied multilevel techniques to graph-based problems such as mesh partitioning [36], the TSP [34], and graph coloring [35]. The basic idea of a multilevel scheme is simple. Starting from the original problem instance, smaller and smaller problem instances are obtained by successive coarsening until some stopping criteria are satisfied. This creates a hierarchy of problem instance in which the problem instance of a given level is always smaller (or of equal size) to the problem instance of the next lower level. Then, a solution is computed to the smallest problem instance and successively transformed into a solution of the next higher level until a solution for the original problem instance is obtained. At each level, the obtained solution might be subject to a refinement process. This idea is illustrated with respect to the application of ACO as refinement process in Figure 4.4. The idea behind the multi-level framework is to solve the problem "globally" in the smallest level, while refining the obtained solution in higher levels.

[4]Note that in standard ACO variants the upper bound of the pheromone values depends on the pheromone update and on the problem instance that is tackled.

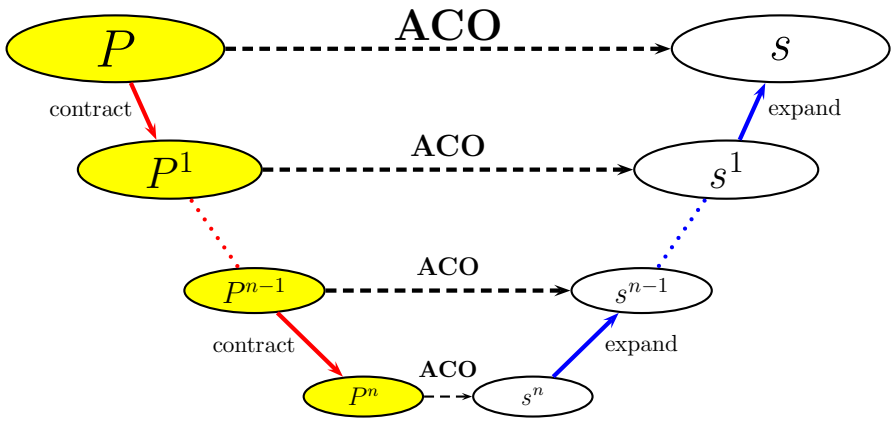


Figure 4.4: ML-ACO: Applying ACO in the multilevel framework. The original problem instance is P. In an iterative process this problem instance is simplified (i.e., contracted) until the lowest level instance P^n is reached. Then, an ACO algorithm (or any other optimization technique) can be used to tackle problem P^n. The obtained best solution s^n is expanded into a solution $s^{n-1'}$ of the next bigger problem instance P^{n-1}. With this solution as the first best-so-far solution, the same ACO algorithm might be applied to tackle problem instance P^{n-1} resulting in a best obtained solution s^{n-1}. This process goes on until the original problem instance was tackled.

Chapter 5

ACO approaches to the SBH

In the previous section we have described the meta-heuristic called ACO. We have also introduced some ACO variants. In this chapter we will propose two different adaptations of ACO algorithms to solve the SBH problem. First a $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) implemented in the hyper-cube framework (HCF) is described. Using this first implementation as a basis, we will propose some changes of the algorithm that produces an Ant Colony System (ACS) also implemented in the HCF. Finally we will introduce a multi-level framework for the problem based on the SM heuristic (see Section 3.5).

5.1 The objective function

Before we outline our particular ACO implementations for SBH, we first deal with an issue concerning the objective function described in Section 2.2.1. Recall that the objective of the SBH problem is to find a directed path p^* in G with $l(p^*) \geq l(p)$ for all possible paths p that fulfil $c(p) \leq n$ (see Equation 2.8). Given a feasible solution p to a problem instance, the original objective function value $l(p)$ is the number of oligonucleotides in p.

This objective function has many plateaus, which results in the following disadvantage when used in a search algorithm. Let p and p' be two solutions with $l(p) = l(p')$ and $c(p) < c(p')$.[1] Even that the objective function $l(\cdot)$ can not distinguish between p and p', the intuition is to prefer p, because the DNA sequence it induces is shorter. This implies a higher chance for an extension of p while respecting the constraint $c(p) \leq n$. Therefore, we define an indirect objective function $f(\cdot)$ as follows:

$$f(p) > f(p') \Leftrightarrow l(p) > l(p') \textbf{ or } \big(l(p) = l(p') \text{ and } c(p) < c(p')\big) \tag{5.1}$$

5.2 The $\mathcal{MM}$AS algorithm

Our first ACO approach is a $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) implemented in the hyper-cube framework (HCF). It solves the SBH problem as shown in Algorithm 9. The data structures used by this algorithm, in addition to counters and to the pheromone model $\mathcal{T}$, are:

- the *iteration-best* solution p_{ib}: the best solution generated in the current iteration by the ants;

[1] Remember that $c(p)$ denotes the length of the DNA sequence derived from p.

Algorithm 9 $\mathcal{MM}$AS for the SBH problem

input: A graph $G = (S, E)$, and the length of the target sequence n
p_{bs} := NULL
p_{rb} := NULL
$cf := 0$
bs_update := FALSE
InitializePheromoneValues($\mathcal{T}$)
while termination conditions not satisfied **do**
  for $j := 1$ to n_f **do**
    p_j := ConstructForwardSolution($\mathcal{T}$)
  end for
  for $j := n_f + 1$ to $n_f + n_b$ **do**
    p_j := ConstructBackwardSolution($\mathcal{T}$)
  end for
  p_{ib} := argmax($f(p_1), ..., f(p_{n_f+n_b})$)
  if p_{rb} = NULL **or** $f(p_{ib}) > f(p_{rb})$ **then** $p_{rb} := p_{ib}$
  if p_{bs} = NULL **or** $f(p_{ib}) > f(p_{bs})$ **then** $p_{bs} := p_{ib}$
  ApplyPheromoneUpdate(*cf*,*bs_update*,$\mathcal{T}$,p_{ib},p_{rb},p_{bs})
  cf := ComputeConvergenceFactor($\mathcal{T}$)
  if $cf > 0.9999$ **then**
    if *bs_update* = TRUE **then**
      ResetPheromoneValues($\mathcal{T}$)
      p_{rb} := NULL
      bs_update := FALSE
    else
      bs_update := TRUE
    end if
  end if
end while
output: DNA sequence s that is obtained from p_{bs}

- the *best-so-far* solution p_{bs}: the best solution generated since the start of the algorithm;
- the *restart-best* solution p_{rb}: the best solution generated since the last restart of the algorithm;
- the *convergence factor* cf, $0 \leq cf \leq 1$: a measure of how far the algorithm is from convergence;
- the Boolean variable *bs_update*: it becomes true when the algorithm reaches convergence for the first time after the beginning of the algorithm or a restart.

The algorithm works as follows. First, all the variables are initialized, and the pheromone values are set to their initial value 0.5 in procedure InitializePheromoneValues($\mathcal{T}$). At each iteration, first n_f ants construct a solution each in procedure ConstructForwardSolution($\mathcal{T}$), and then n_b ants construct a solution each in procedure ConstructBackwardSolution($\mathcal{T}$). A *forward solution* is constructed from left to right, and a *backward solution* from right to left.

Subsequently, the value of the variables p_{ib}, p_{rb} and p_{bs} is updated (note that, until the first restart of the algorithm, it holds that $p_{rb} \equiv p_{bs}$). Fourth, pheromone values are updated via the ApplyPheromoneUpdate(cf, *bs_update*, $\mathcal{T}$, p_{ib}, p_{rb}, p_{bs}) procedure. Fifth, a new value for the convergence factor cf is computed. Depending on this value, as well as on the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm or not is taken. If the algorithm is restarted, the procedure ResetPheromoneValues($\mathcal{T}$) is applied and all the pheromones are reset to their initial value (0.5). The algorithm is iterated until some opportunely defined termination conditions are satisfied. Once terminated the algorithm returns the best-so-far solution p_{bs}. The main procedures of Algorithm 9 are now described in detail.

ConstructForwardSolution($\mathcal{T}$)

Starting from an empty path $p = \langle\rangle$, this function constructs a path in G from left to right by adding exactly one oligonucleotide at each construction step. This is done probabilistically using a pheromone model $\mathcal{T}$, which consists of pheromone values $\tau_{s,s'}$ and $\tau_{s,s'}$ for each pair $s, s' \in S$ $(s \neq s')$, that is, to each directed link of G is associated a pheromone value. Additionally, $\mathcal{T}$ comprises pheromone values $\tau_{s_0,s}$ and τ_{s,s_0} for all $s \in S$, where s_0 is a non-existing dummy oligonucleotide. This procedure is also defined in Algorithm 10.

Given the current path $p = \langle p[1], \ldots, p[l(p)]\rangle$, $\hat{S} = S \setminus \{p[1], \ldots, p[l(p)]\}$ is the set of available oligonucleotides, that is, the set of oligonucleotides that can be added to p at the next construction step. Such a construction step is performed as follows. First, we compute a desirability value

$$\mu_{p[l(p)],s} := [\tau_{p[l(p)],s}]^{\alpha} \cdot [\eta_{p[l(p)],s}]^{\beta}$$

for all $s \in \hat{S}$, where

$$\eta_{p[l(p)],s} := \frac{o_{p[l(p)],s}}{(l-1)},$$

$\alpha = 1$ and $\beta = 5$ (β has been set to 5 in order to give a high heuristic guidance to the algorithm at the start of the search). The values $\eta_{p[l(p)],s}$ are called *heuristic information*. They are defined such that $\eta_{p[l(p)],s} \in [0, 1]$ grows with growing overlap $o_{p[l(p)],s}$ between the oligonucleotides $p[l(p)]$ and s. Note that when the pheromone values are all equal, the desirability value $\mu_{p[l(p)],s}$ is high exactly when $o_{p[l(p)],s}$ is high. Then, we generate a so-called *restricted candidate list* $\hat{S}^{\text{rcl}} \subseteq \hat{S}$ with a pre-defined cardinality *cls* such that $\mu_{p[l(p)],s} \leq \mu_{p[l(p)],u}$ for all $s \in \hat{S} \setminus \hat{S}^{\text{rcl}}$ and $u \in \hat{S}^{\text{rcl}}$. Then, with probability $q \in [0, 1)$ the next oligonucleotide $p[l(p)+1]$ is chosen from $\hat{S}^{\text{rcl}}$ such that

$$p[l(p)+1] := \arg\max_{s \in \hat{S}} \{\mu_{p[l(p)],s}\} \ . \tag{5.2}$$

Otherwise, the next oligonucleotide $p[l(p)+1]$ is chosen from $\hat{S}^{\text{rcl}}$ by roulette-wheel-selection according to the following probabilities:

$$\mathbf{p}_{p[l(p)],s} := \frac{\mu_{p[l(p)],s}}{\sum_{u \in \hat{S}^{\text{rcl}}} \mu_{p[l(p)],u}} \tag{5.3}$$

Note that q (henceforth called the determinism rate) and *cls* are important parameters of the algorithm.

Algorithm 10 ConstructForwardSolution method of Algorithm 9

input: A graph $G = (S, E)$, a pheromone trail $\mathcal{T}$, and the length of the target sequence n
$\hat{S} := S$
$\hat{S}^{\text{rcl}} :=$ the set of size cls of $s \in \hat{S}$ which maximize $\mu_{0,s}$
$r :=$ random number from [0,1]
$pr_{\text{sum}} := 0$
for $s \in \hat{S}^{\text{rcl}}$ **do**
  $\mathbf{p}_{s_0,s} := \mu_{s_0,s} / \sum_{u \in \hat{S}^{\text{rcl}}} \mu_{s_0,u}$
  $pr_{\text{sum}} := pr_{\text{sum}} + \mathbf{p}_{s_0,s}$
  if $pr_{\text{sum}} \geq r$ **then**
    $s^* := s$
    end loop
  end if
end for
$p := \langle s^* \rangle$
while $c(p) < n$ **do**
  $\hat{S} := \hat{S} \setminus \{s^*\}$
  $r :=$ random number from [0,1]
  if $r < q$ **then**
    $s^* := \text{argmax}_{s \in \hat{S}} \{\mu_{p[l(p)],s}\}$
  else
    $\hat{S}^{\text{rcl}} :=$ the set of size cls of $s \in \hat{S}$ which maximize $\mu_{p[l(p)],s}$
    $r :=$ random number from [0,1]
    $pr_{\text{sum}} := 0$
    for $s \in \hat{S}^{\text{rcl}}$ **do**
      $\mathbf{p}_{p[l(p)],s} := \mu_{p[l(p)],s} / \sum_{u \in \hat{S}^{\text{rcl}}} \mu_{p[l(p)],u}$
      $pr_{\text{sum}} := pr_{\text{sum}} + \mathbf{p}_{p[l(p)],s}$
      if $pr_{\text{sum}} \geq r$ **then**
        $s^* := s$
        end loop
      end if
    end for
  end if
  Extend path p by adding s^* to its end
end while
output: $p :=$Find_Best_Subpath(p)

The only construction step that is different is the first one, that is, when $p = \langle\rangle$. In this case, the desirability values are computed as

$$\mu_{s_0,s} := \tau_{s_0,s}^{\alpha} \cdot [\eta_{s_0,s}]^{\beta} \in [0, 1]$$

for all $s \in \hat{S}$ (note that $\hat{S} = S$ when $p = \langle\rangle$). Hereby,

$$\eta_{s_0,s} := \frac{l - o_{\text{bpre}(s),s} + o_{s,\text{bsuc}(s)}}{2(l-1)} \quad , \tag{5.4}$$

Functions bpre(s) and bsuc(s) are defined in equations 3.1 and 3.2. Remember that we call bpre(s) the best predecessor of s, that is, the oligonucleotide that has the highest overlap with s when placed before s, and we call bsuc(s) the best successor of s. In both cases, if there is more than one best predecessor (respectively, successor), the first one found is taken. Note that this way of defining the heuristic information favors oligonucleotides that have a very good "best successor", and at the same time a bad "best predecessor". The intuition is that the spectrum most probably does not contain an oligonucleotide that is a good predecessor for the first oligonucleotide of the target sequence. The constants $\alpha = 1$ and $\beta = 5$ have the same value as before.

Having defined the desirability value for the first construction step, the further procedure concerning the derivation of the restricted candidate list $\hat{S}^{\text{rcl}}$ and the choice of one of the oligonucleotides from $\hat{S}^{\text{rcl}}$ is the same as outlined above for standard construction steps.

Finally, the solution construction stops as soon as $c(p) \geq n$, that is, when the DNA sequence derived from the constructed path p is at least as long as the target sequence s_t. In case $c(p) > n$, we look for the longest (in terms of the number of oligonucleotides) subpath p' of p such that $c(p) \leq n$, and replace p by p' (see function Find_Best_Subpath(p) in Section 3.1).

ConstructBackwardSolution($\mathcal{T}$)

This function works principally in the same way as function ConstructForwardSolution($\mathcal{T}$). The first difference is that a solution p is constructed from right to left. The second difference is that—given a partial solution p —the desirability values are still computed as if the solution construction were from left to right. For example, the desirability value of adding an oligonucleotide s to the front of p is $\mu_{s,p[1]}$ (instead of $\mu_{p[1],s}$). This is done such that for the construction of a solution p the same pheromone values are used, no matter if the solution is constructed from left to right, or from right to left.

ApplyPheromoneUpdate(**cf,bs_update,**$\mathcal{T}$**,**p_{ib}**,**p_{rb}**,**p_{bs}**)**

As usual for $\mathcal{MM}$AS implementations in the HCF, we use at each iteration a weighted combination of the solutions p_{ib}, p_{rb}, and p_{bs} for updating the pheromone values. The weight of each solution depends on the value of the convergence factor cf and on the Boolean variable *bs_update*. In general, the pheromone update is performed as follows:

$$\tau_{s,u} := \tau_{s,u} + \rho \cdot (m_{s,u} - \tau_{s,u}) \ \ ,\forall\, \tau_{s,u} \in \mathcal{T} \ \ , \tag{5.5}$$

where $\rho \in (0, 1]$ is a constant called learning rate, and $m_{s,u}$ is composed as follows:

$$m_{s,u} := (\kappa_{ib} \cdot \delta_{s,u}(p_{ib})) + (\kappa_{rb} \cdot \delta_{s,u}(p_{rb})) + (\kappa_{bs} \cdot \delta_{s,u}(p_{bs})) \ \ , \tag{5.6}$$

where κ_{ib} is the weight of solution p_{ib}, κ_{rb} is the weight of solution p_{rb}, κ_{bs} is the weight of solution p_{bs}, and $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. Moreover, when $s \neq s_0$ and $u \neq s_0$, $\delta_{s,u}(p)$ is a function that returns 1 in case u is the direct successor of s in p, and 0 otherwise. In case $s = s_0$, $\delta_{s_0,u}(p)$ returns 1 in case u is the first oligonucleotide in p, and 0 otherwise. In case $u = s_0$, $\delta_{s,s_0}(p)$ returns 1 in case s is the last oligonucleotide in p, and 0 otherwise. After the pheromone update rule (Equation 5.5) is applied, pheromone values that exceed an upper limit of $\tau_{max} = 0.99$ are set back to τ_{max}, and pheromone values that fall below a lower limit $\tau_{min} = 0.01$ are set back to τ_{min}. This prevents the algorithm from complete convergence.

Table 5.1: Setting of κ_{ib}, κ_{rb} and κ_{bs} depending on the convergence factor cf and the Boolean control variable *bs_update*.

	bs_update = FALSE				*bs_update*
	$cf < 0.7$	$cf \in [0.7, 0.9)$	$cf \in [0.9, 0.95)$	$cf \geq 0.95$	= TRUE
κ_{ib}	1	2/3	1/3	0	0
κ_{rb}	0	1/3	2/3	1	0
κ_{bs}	0	0	0	0	1

Equation 5.6 allows to choose how to schedule the relative influence of the three solutions used for updating pheromones. The exact schedule for the setting of the three solution weights used by $\mathcal{MM}$AS in the HCF is shown in Table 5.1. In the early stages of the search (i.e., when $cf < 0.7$), only the iteration-best solution is used. Then, when the value of the convergence factor increases (i.e., $0.7 \leq cf < 0.9$) one third of the total influence is given to the restart-best solution, which then increases to two thirds when $0.9 \leq cf < 0.95$. Eventually, all the influence is given to the restart-best solution (i.e., when $cf \geq 0.95$). Once the value of the convergence factor raises above 0.9999, the Boolean control variable *bs_update* is set to TRUE, and all the influence is given to the best-so-far solution. Note that all these values and limits were chosen after a careful tuning by hand.

ComputeConvergenceFactor($\mathcal{T}$)

The convergence factor cf, which is a function of the current pheromone values, is computed as follows:

$$cf := 2 \left(\left(\frac{\sum_{\tau_{s,u} \in \mathcal{T}} \max\{\tau_{\max} - \tau_{s,u}, \tau_{s,u} - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right)$$

This formula says that when the algorithm is initialized (or reset) so that all pheromone values are set to 0.5, then $cf = 0$, while when the algorithm has converged, then $cf = 1$. In all other cases, cf has a value in $[0, 1]$.

5.3 The ACS algorithm

Our second ACO approach is an Ant Colony System (ACS) implemented in the hyper-cube framework (HCF). It solves the SBH problem as shown in Algorithm 11. This algorithm has many parts in common with the $\mathcal{MM}$AS described in previous section.

In terms of data structures—in addition to counters and the pheromone model $\mathcal{T}$—, the algorithm only uses the best-so-far solution (i.e., p_{bs}) for updating the pheromone values. Remember that p_{bs} is the best solution generated by the ants since the start of the algorithm.

The general procedure of the ACS algorithm is as follows. First, all the variables are initialized, and pheromone values are set to their initial value 0.5 in procedure InitializePheromoneValues($\mathcal{T}$). At each iteration, first n_f ants construct a solution each in procedure ConstructForwardSolution($\mathcal{T}$), and then n_b ants construct a solution each in procedure ConstructBackwardSolution($\mathcal{T}$). After the generation of a solution p_j the method ReducePheromone($\mathcal{T}$,p_j) is called

which updates the pheromone trails according to the sequence of nodes in p_j, decreasing the pheromone values. Finally, the global pheromone update is performed in function ApplyPheromoneUpdate($\mathcal{T}$,p_{bs}). The algorithm is iterated until some opportunely defined termination conditions are satisfied. Once terminated the algorithm returns the best-so-far solution p_{bs}. The main procedures of Algorithm 11 are now described in detail.

Algorithm 11 ACS for the SBH problem

```
1: input: A graph G = (S, E), and the length of the target sequence n
2: p_bs := NULL
3: InitializePheromoneValues(T)
4: while termination conditions not satisfied do
5:    for j := 1 to n_f do
6:       p_j := ConstructForwardSolution(T)
7:       ReducePheromone(T,p_j)
8:    end for
9:    for j := n_f + 1 to n_f + n_b do
10:      p_j := ConstructBackwardSolution(T)
11:      ReducePheromone(T,p_j)
12:   end for
13:   if p_bs = NULL then
14:      p_bs := argmax{f(p_1), ..., f(p_{n_f+n_b})}
15:   else
16:      p_bs := argmax{f(p_1), ..., f(p_{n_f+n_b}), f(p_bs)}
17:   end if
18:   ApplyPheromoneUpdate(T,p_bs)
19: end while
20: output: p_bs
```

ConstructForwardSolution($\mathcal{T}$) and ConstructBackwardSolution($\mathcal{T}$) are the same as in the $\mathcal{MM}$AS algorithm. For their explanation we refer to previous section.

ReducePheromone($\mathcal{T}$,p_j)

This function updates the pheromone trails by decreasing the pheromone in edges used in path p_j as described in section 4.3.1. This update is applied to all $\tau_{p_j[i],p_j[i+1]},\ i \in [1, l(p_j)-1]$ (i.e., to all the pheromone values representing used edges of path p_j), to $\tau_{s_0,p[0]}$ and to $\tau_{p[l(p)],s_0}$ (i.e., the pheromone to determine the first and the last sequence):

$$\tau_{s,u} := (1-\xi)\cdot\tau_{s,u} + \xi\cdot\tau_{\min}, \tag{5.7}$$

where $\tau_{\min}$ is a small positive constant such that $0 \leq \tau_{\min} \leq 0.5$, and $\xi \in [0,1]$ is a value close to zero. In this work we will use $\tau_{\min} = 0.01$.

ApplyPheromoneUpdate($\mathcal{T}$,p_{bs})

This method is different from the $\mathcal{MM}$AS method because it only uses the best-so-far solution to update the pheromone model. It also differs from a usual ACS pheromone update because

it updates all the pheromone values (instead of updating only the solution components in p_{bs}). Therefore each value of $\mathcal{T}$ is updated according to the following formula:

$$\tau_{s,u} := \tau_{s,u} + \rho \cdot (\delta_{s,u}(p_{bs}) - \tau_{s,u}) , \quad \forall \tau_{s,u} \in \mathcal{T} ,$$

where $\rho \in (0, 1]$ is the learning rate. Moreover, as defined in $\mathcal{MM}$AS, when $s \neq s_0$ and $u \neq s_0$, $\delta_{s,u}(p)$ is a function that returns 1 in case u is the direct successor of s in p, and 0 otherwise. In case $s = s_0$, $\delta_{s_0,u}(p)$ returns 1 in case u is the first oligonucleotide in p, and 0 otherwise. In case $j = s_0$, $\delta_{s,s_0}(p)$ returns 1 in case s is the last oligonucleotide in p, and 0 otherwise.

5.4 The multi-level framework

In section 3.5 we proposed a constructive heuristic called *Sub-sequence Merger (SM)* for SBH. We will use the same idea to define a multi-level framework in which the ACO algorithms outlined in the previous sections can be applied.

5.4.1 Instance contraction

Algorithm 12 Instance contraction

1: **input:** a problem instance $(G = (S, E), n)$
2: $P \leftarrow \{\langle s \rangle \mid s \in S\}$
3: *stop* := FALSE
4: *level* := 1
5: **for** *overlap* := $l - 1, \ldots, 1$ **do**
6: *changed* := FALSE
7: **while** $\exists\, p, p' \in P$ **s.t.** $o_{pp'} \geq$ *overlap* **&** $|S_{\text{bsuc}}(p)| = 1$ **&** $|S_{\text{bpre}}(p')| = 1$ **&** $\text{bsuc}(p) = p'$ **&** $\text{bpre}(p') = p$ **&** **not** *stop* **do**
8: *changed* := TRUE
9: Add path p' to the end of path p
10: $P := P \setminus \{p'\}$
11: **if** $c(p) \geq n$ **then**
12: *stop* := TRUE
13: **end if**
14: **end while**
15: **if not** *stop* **and** *changed* **then**
16: $(G^{level}, n) :=$ GenerateProblemInstance(P)
17: *level* := *level* + 1
18: **end if**
19: **end for**
20: **output:** A sequence of instances $(G^0, n), (G^1, n), \ldots, (G^d, n)$

The first step of a multi-level framework consists in contracting the original problem instance iteratively in order to generate a sequence of smaller and smaller problem instances. In the case of the SBH problem we use the following contraction mechanism (see also Algorithm 12): At each contraction step we have given a set P of paths in G (in fact, the

contraction starts from a set of $|S|$ paths, each of which contains exactly one oligonucleotide $i \in S$). A contraction step consists of merging some of these paths. Hereby, we consider only those paths p and p' where the last oligonucleotide of p and the first one of p' have a fixed overlap, which is different for each construction step; starting from the maximum $l-1$ and getting reduced step by step. In addition it is required that p' is the unique best successor of p, and that p is the unique best predecessor of p'. The best successor (i.e., bpre(p)) and the best predecessor (i.e., bpre(p)) of a path p are defined in equations 3.7. Furthermore, in Algorithm 12 $S_{\text{bsuc}}(p)$ is defined as the set of best successors of p, that is,

$$S_{\text{bsuc}}(p) := \{p' \in P \mid o_{p,p'} = o_{p,\text{bsuc}(p)}\};$$

and $S_{\text{bpre}}(p)$ is defined as the set of best predecessors of p, that is,

$$S_{\text{bpre}}(p) := \{p' \in P \mid o_{p',p} = o_{\text{bpre}(p),p}\}.$$

The idea of this restriction is produce possibly error free sub-sequences of the original DNA target sequence.

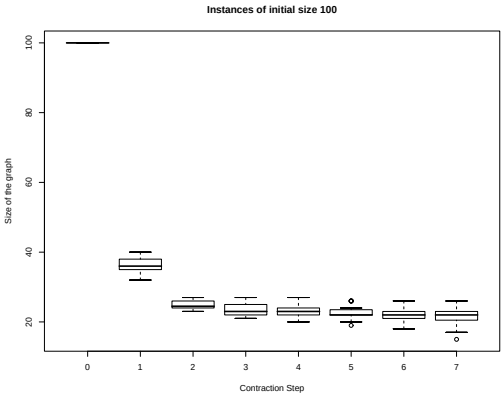
Each contraction step leads to a new set of paths P from which a new (smaller) problem instance is generated in function GenerateProblemInstance(P). This is done by deriving from each path $p \in P$ the corresponding DNA strand (as exemplary shown in Figure 2.2(c)). This mechanism generates a sequence $(G^0, n), (G^1, n), \ldots, (G^d, n)$ of smaller and smaller problem instances, where $(G^0, n) \equiv (G, n)$. (G^d, n) denotes the smallest instances that can be obtained Note that a solution to any of these instances can directly be seen as a solution to any of the other instances.

In order to see how the contraction mechanism reduces the size of an instance we have applied the method to all the instances by Błażewicz et al. [3] (for a description of the instances see Section 3.8). We have drawn 5 box-plots in Figures 5.1 and 5.2 that represent the size of the generated instances by the contraction algorithm. As the number of contraction steps depends on the instance, in the x axis, we have placed the value of the variable *overlap* of Algorithm 12 instead of the step; to each value x of the axis corresponds the size of the instance G^{level} in line 15 of algorithm 12 when the *overlap* was equal to x.

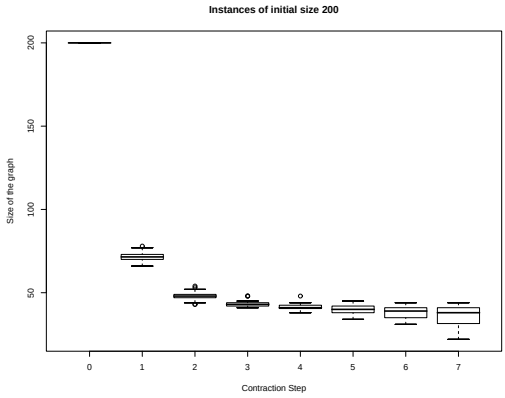
From the graphics we can draw the following conclusions. First, the size of the original problem G^0 is much bigger than the size of the first level G^1. Generally, the first contraction step produces a problem instance that is about 40% of the size of the original instance. Second, the second contraction step does not reduce the size of the instance as much as the first step. The size of the second level G^2 is about 60% of the size of the first level G^1. Third, generally speaking, the next contraction levels do not decrease the size of the instance significantly, specially in small size instances. As a general conclusion, we can say that each step reduces the instance in a smaller factor than its predecessor level.

5.4.2 Application of ACO in the multi-level framework

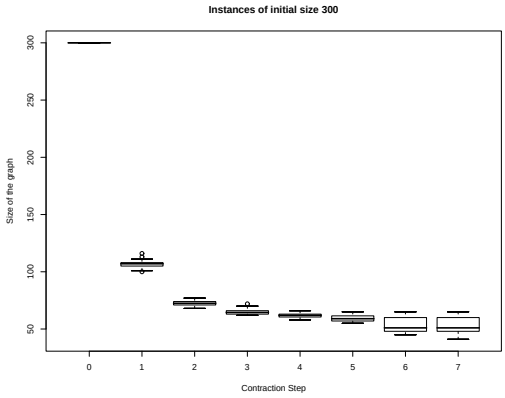
The application of the ACO algorithm proposed in Section 5.2 in the multi-level framework (ML-ACO) works as follows. Given the sequence $I = \langle (G^0, n), (G^1, n), \ldots, (G^d, n) \rangle$ of problem instances, ACO is first applied to the smallest instance (G^d, n). Subsequently, ACO is applied in the given order to all problem instances $(G^{d-1}, n), \ldots, (G^0, n)$. Hereby we always use the best solution of the ACO algorithm found for an instance (G^{r-1}, n) as first best-so-far solution for the application of ACO to the instance (G^r, n). As stopping condition for the



(a) Contraction for instances of size $n = 100$

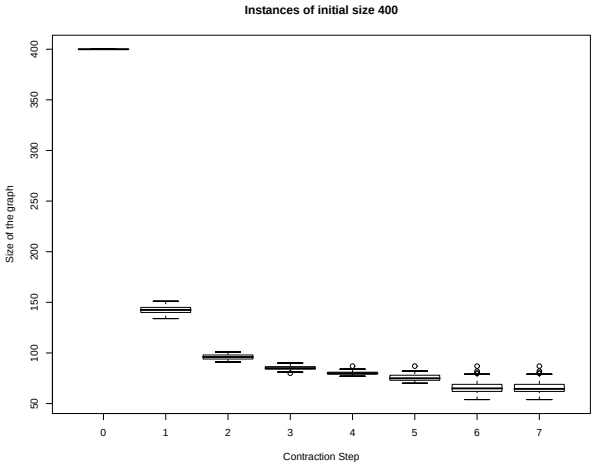


(b) Contraction for instances of size $n = 200$

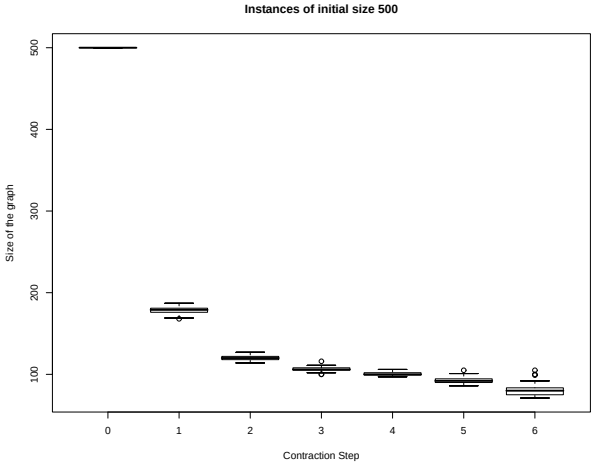


(c) Contraction for instances of size $n = 300$

Figure 5.1: Size of the instances generated in every step of the contraction method for instances by Błażewicz et al. [3] of size 100 to 300. For each contraction step we show the distribution of the resulting instance size (over 40 instances).



(a) Contraction for instances of size $n = 400$



(b) Contraction for instances of size $n = 500$

Figure 5.2: Size of the instances generated in every step of the contraction method for instances by Błażewicz et al. [3] of size 400 and 500. For each contraction step we show the distribution of the resulting instance size (over 40 instances).

whole procedure we use a CPU time limit. The given CPU time is distributed such that the application of ACO to an instance (G^r, n) is always allocated a constant factor $f_{time} \geq 1$ of the CPU time that is allocated for the application of ACO to instance (G^{r-1}, n). Thus, let t_r be the amount of time allocated for the instance (G^r, n), the time allocated for instance (G^{r-1}, n) will be $t_{r-1} = t_r \cdot f_{time}$. Due to the fact that instance (G^{r-1}, s_t) is smaller than instance (G^r, s_t) it is reasonable to allocate more time to (G^r, s_t).

Algorithm 13 ACO in the multilevel framework

```
1: input: A graph G, and the length of the target sequence n
2: I := the set of instances created with algorithm 12
3: p_bs := NULL
4: for i := d, ..., 0 do
5:    p := Apply an ACO algorithm to (G^i, n); introducing p_bs as the best solution found,
      until it stops
6:    if f(p) > f(p_bs) then
7:       p_bs := p
8:    end if
9: end for
10: output: DNA sequence s that is obtained from p_bs
```

For the application of ACO to an instance (G^r, s_t) we use two stopping conditions: (1) the allocated CPU time, and (2) the maximum number of iterations it_{wb} without improving the best-so-far solution. Whenever one of the two conditions is fulfilled the application of ACO at the corresponding level is terminated, and the application to the next level starts. Note that the use of the second stopping condition implies that the last application of ACO (that is, the application to the original instance (G^0, s_t)) may use all the remaining CPU time, which is sometimes more than the allocated CPU time. Moreover, the second stopping condition is not used for the last application of ACO.

Chapter 6

Results of the ACO algorithms

In the last chapter we proposed some ACO implementations to solve the SBH problem. However all these algorithms have some parameters for which we have not yet defined a value. In this chapter we will first present various tests of the algorithms with different configurations in order to define which values are the best for the parameters. Later we will study the results of the algorithms with the tuned parameters.

We implemented the 3 ACO variants in ANSI C++ using GCC 3.2.2 for compiling the software. Our experimental results were obtained on a PC with AMD64X2 4400 processor and 4 Gb of memory. We have used the instances described in Section 3.8 to tune the algorithms. Each algorithm was applied with each configuration 10 times to each problem instance in order to minimize the effect of randomness in the results. Furthermore, for each run we have allocated a computation time limit to the algorithms depending on the size of the instances (see Table 6.1).

Table 6.1: Allocated computation time limits

Size n of the instance	Allocated time in seconds
109	2
209	10
309	50
409	100
509	200

When we display the results for the different executions we will use—as in Chapter 3—the following characteristics to compare them:

- The solution quality, that is the number of oligonucleotides in the constructed paths. Remember that the optimization objective in the SBH problem is to maximize this value.
- The number of solved problem instances, that is, the number of instances for which a path of maximal length could be found.[1]

[1]Remember in this context that an optimal solution to the SBH problem does not necessarily correspond to a DNA sequence that is equal to the target sequence.

- The similarity scores obtained by comparing the computed DNA sequence with the DNA target sequence. We will use both average scores obtained from the Needleman-Wunsch algorithm [29], which is an algorithm for global alignment and, in contrast, the average scores obtained by the application of the Smith-Waterman algorithm, which is an algorithm for local alignment. Both algorithms have been applied with the following parameters: +1 for a match of oligonucleotides, -1 for a mismatch or a gap.
- Finally, we will display the computation time that it takes to the algorithm to solve one instance (in seconds). We consider that the time used for the algorithm is the time that it takes to find the best solution found (i.e., the returned solution), however, in case that no optimal solution is found, the real execution time of the algorithm may be higher.

6.1 The tuning

In this section we will first test different configurations for the different ACO algorithms. These algorithms have many parameters to be tested. However, due to the exponential grow of the number of configurations to be tested in respect to the number of parameters to be tuned, some of the parameters have been fixed in all the executions. The value of these parameters has been chosen after a careful tuning by hand. We don't necessarily expect this values to be the best possible values, but we believe them to be reasonable. In all the ACO algorithms ρ has been set to 0.1, α to 1, and β to 5.

6.1.1 $\mathcal{MM}$AS tuning

In the tuning of $\mathcal{MM}$AS the parameters which have been tuned are n_f and n_b (i.e., the number of forward and backward ants), cls (i.e.,the size of the restricted candidate list), and q (the determinism rate). Table 6.2 shows the possible values assigned to each tuned parameter.

Table 6.2: Tuning values for the $\mathcal{MM}$AS algorithm

Variable	value
$\{n_f, n_b\}$	$\{0,6\}$,$\{3,3\}$,$\{6,0\}$
cls	2, 3, 5, 10, all
q	0.0, 0.5, 0.75, 0.9, 0.95

We have tested all the possible combinations of all the parameters; therefore 75 different configurations have been tested. Figures 6.1 to 6.5 show the results obtained in all the executions. Each of the figures shows the average results of the executions of all the configurations for each length of the target sequence (i.e., one figure for each value of n). Each figure contains nine matrices in three rows and three columns. Each row of matrices contains the results of one configuration of n_f, n_b (i.e., the number of forward and backward ants). The first matrix of each row (the first column of matrices) corresponds to the results of the global similarity score obtained by comparing the target sequence s_t and the obtained sequence (using the Needleman-Wunsch algorithm [29]). The second matrix corresponds to the number of solved

instances (i.e. when $l(p)$ is optimal). We have considered that an instance was solved when at least in five (out of ten) trials the instance was solved. This was done in order to distinguish better between the configurations. The last matrix of each sub-figure corresponds to the computation time. Each matrix contains five rows and five columns. Rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all"[2] from top to bottom) and the columns correspond to the values of determinism q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). In the matrix which represents the number of solved instances, the values are displayed for each configuration in numerical form. For an easier read, for both time and score matrices, we have translated the values into gray scale: the best value obtained in all configurations received gray value 1.0 (i.e., white) and the worst received gray value 0.0 (i.e, black). All the other values received a gray value proportional to the distance between the best value and the worst value. For example, a value in the middle between the best and the worst would get a gray value of 0.5 (i.e., medium gray). Therefore, in the score matrix—the first one of each sub-figure—, light grays represent high values and dark grays low values whereas in the time matrices—the third one of each sub-figure—light grays represent low values and dark grays high values. The numerical values of white and black can be found in the caption of the figures.

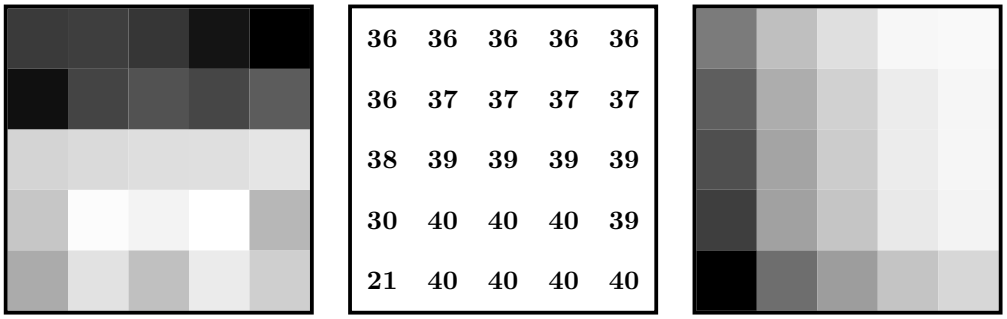
From the results that are displayed in Figures 6.1 to 6.5 we can draw the following conclusions. First, the use of forward and backward ants, instead of using only ants of one type, improves considerably the results and the running times of the algorithm. The improvement is specially shown in the largest instances (see figure 6.5). Therefore we can consider that it is beneficial to use forward and backward ants at the same time.

Second, lower values of cls improve the efficiency of the algorithm (see Figure 6.1). However, when cls increases, the effectivity (both concerning similarity score and number of solved instances) tends to increase. This conclusion is not applicable to cls = "all" where the results are bad. We can conclude that increasing the size of the candidate list increases the search space enabling to find better solutions, but causing an increase in the computation time needed. When cls is too big, the algorithm does not converge (in the desired time), and gets bad results.

Third, big values of q also improve the efficiency but high values of q tend to decrease effectivity; the effect of q is contrary to the effect of cls. Therefore, a combination of big values of q with small ones of cls decrease the execution time, but the algorithm may not find an optimal solution. The reason of this effect is that this configuration causes ACO to converge fast, therefore good solutions will be computed in short time. However, fast convergence causes that many areas of the search space are not explored which may result in sub-optimal results.

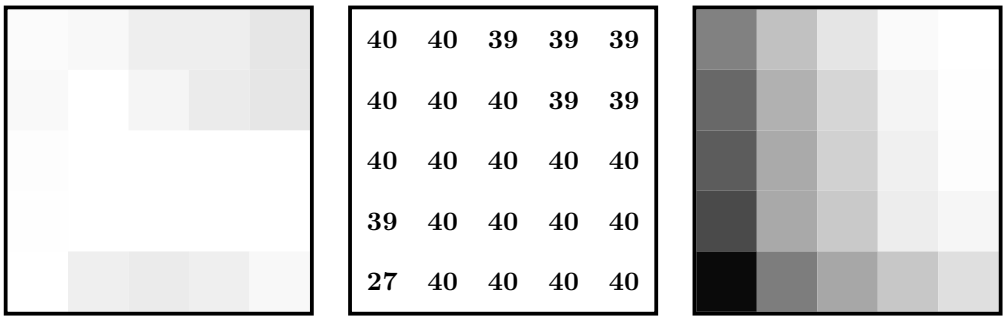
Finally, in terms of similarity score and number of solved instances, there is a region where values are usually high. This region is formed by cls between 5 and 10, and determinism q from 0.5 to 0.9, except for the configuration $cls = 5$ and $q = 0.9$ which we don't consider to be in this region. From this region the fastest configurations are $cls = 5$ and $q = 0.75$ and $cls = 10$ and $q = 0.9$. Therefore these two configurations—with $n_l = 3$ and $n_r = 3$—can be seen as the best configurations for $\mathcal{MM}$AS. In the rest of the work when running the $\mathcal{MM}$AS algorithm we will use the configuration $\rho = 0.1$, $n_l = 3$, $n_r = 3$, $cls = 10$ and $q = 0.9$.

[2] "All" means that the candidate list is not used.



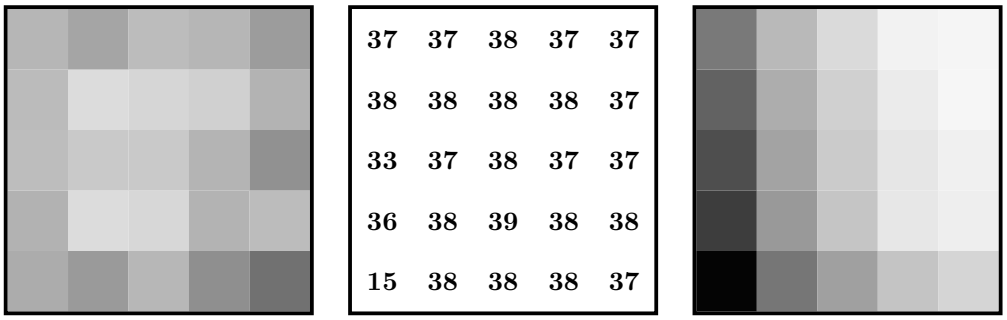
36	**36**	**36**	**36**	**36**
36	**37**	**37**	**37**	**37**
38	**39**	**39**	**39**	**39**
30	**40**	**40**	**40**	**39**
21	**40**	**40**	**40**	**40**

(a) Results for 0 left ants and 6 right ants.



40	**40**	**39**	**39**	**39**
40	**40**	**40**	**39**	**39**
40	**40**	**40**	**40**	**40**
39	**40**	**40**	**40**	**40**
27	**40**	**40**	**40**	**40**

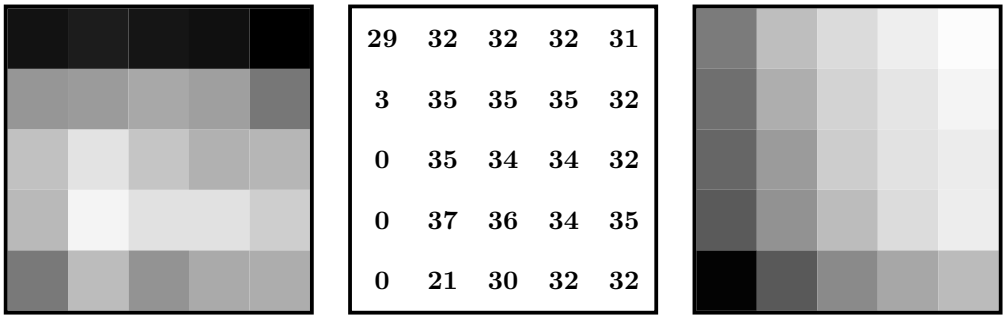
(b) Results for 3 left ants and 3 right ants.



37	**37**	**38**	**37**	**37**
38	**38**	**38**	**38**	**37**
33	**37**	**38**	**37**	**37**
36	**38**	**39**	**38**	**38**
15	**38**	**38**	**38**	**37**

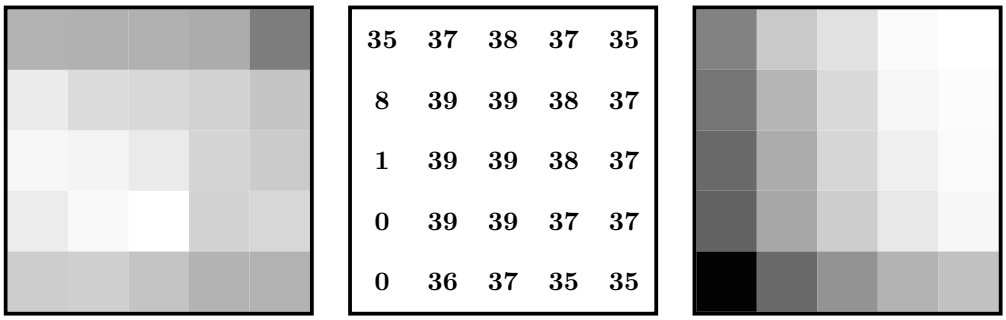
(c) Results for 6 left ants and 0 right ants.

Figure 6.1: Results of the tuning of $\mathcal{MM}$AS concerning the instances with target sequence length $n = 109$. In each of the matrices the rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all" from top to bottom) and the columns correspond to the values of the determinism rate q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 104.117 corresponds to black color, and 108.4 to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 1.464 corresponds to black color, and 0.032 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.



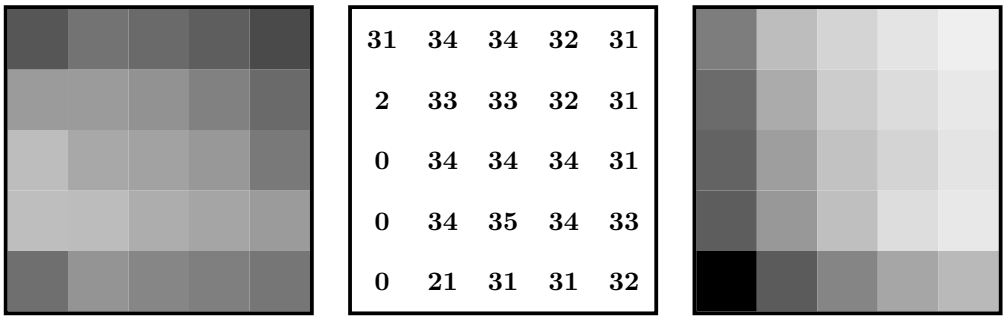
29	32	32	32	31
3	35	35	35	32
0	35	34	34	32
0	37	36	34	35
0	21	30	32	32

(a) Results for 0 left ants and 6 right ants.



35	37	38	37	35
8	39	39	38	37
1	39	39	38	37
0	39	39	37	37
0	36	37	35	35

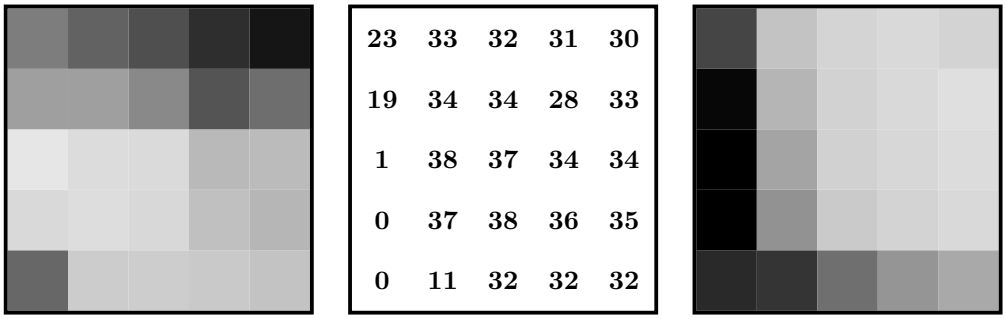
(b) Results for 3 left ants and 3 right ants.



31	34	34	32	31
2	33	33	32	31
0	34	34	34	31
0	34	35	34	33
0	21	31	31	32

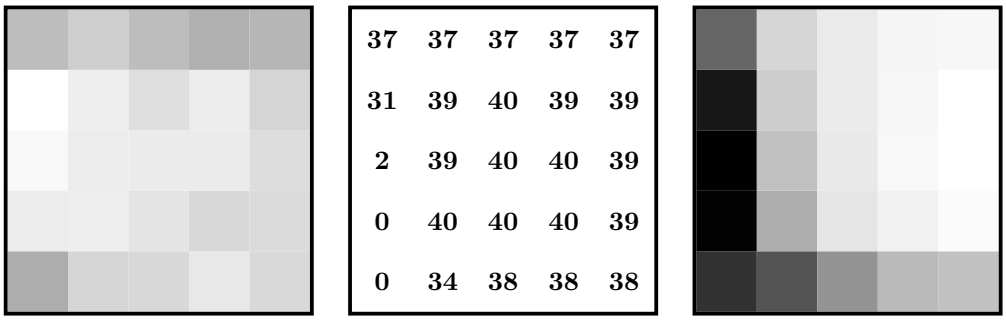
(c) Results for 6 left ants and 0 right ants.

Figure 6.2: Results of the tuning of $\mathcal{MM}$AS concerning the instances with target sequence length $n = 209$. In each of the matrices the rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all" from top to bottom) and the columns correspond to the values of the determinism rate q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 177.533 corresponds to black color, and 206.675 to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 8.839 corresponds to black color, and 0.968 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.



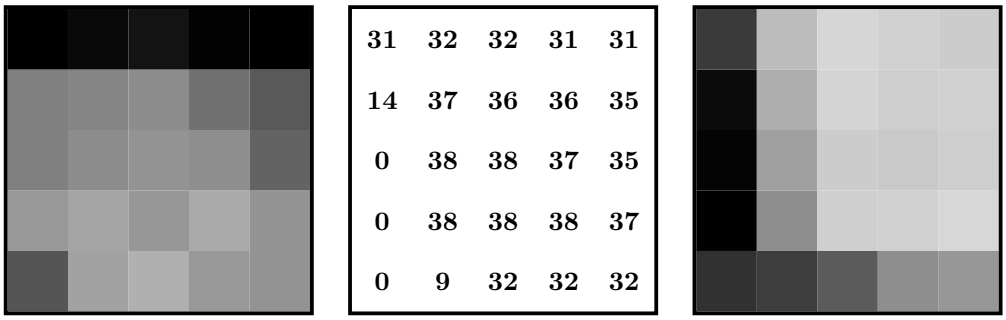
23	**33**	**32**	**31**	**30**
19	**34**	**34**	**28**	**33**
1	**38**	**37**	**34**	**34**
0	**37**	**38**	**36**	**35**
0	**11**	**32**	**32**	**32**

(a) Results for 0 left ants and 6 right ants.



37	**37**	**37**	**37**	**37**
31	**39**	**40**	**39**	**39**
2	**39**	**40**	**40**	**39**
0	**40**	**40**	**40**	**39**
0	**34**	**38**	**38**	**38**

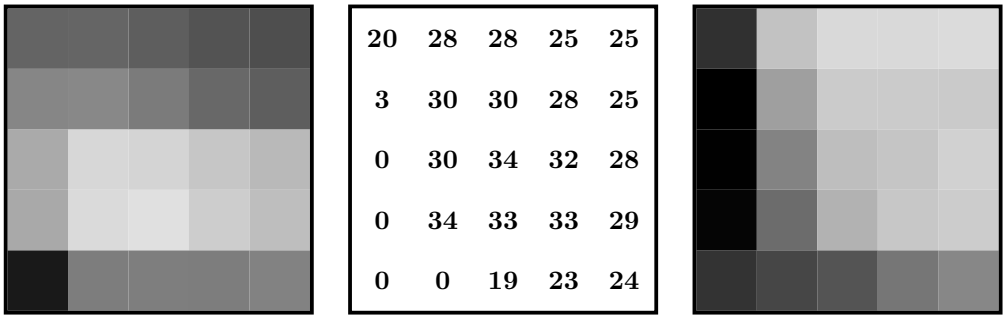
(b) Results for 3 left ants and 3 right ants.



31	**32**	**32**	**31**	**31**
14	**37**	**36**	**36**	**35**
0	**38**	**38**	**37**	**35**
0	**38**	**38**	**38**	**37**
0	**9**	**32**	**32**	**32**

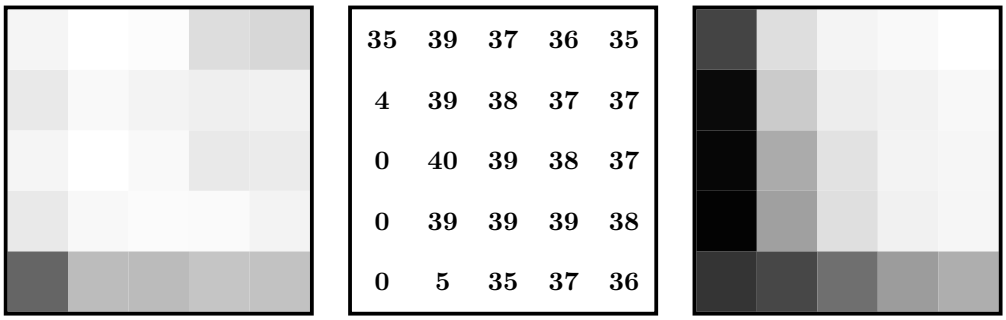
(c) Results for 6 left ants and 0 right ants.

Figure 6.3: Results of the tuning of $\mathcal{MM}$AS concerning the instances with target sequence length $n = 309$. In each of the matrices the rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all" from top to bottom) and the columns correspond to the values of the determinism rate q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 255.187 corresponds to black color, and 305.64 to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 38.043 corresponds to black color, and 3.604 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.



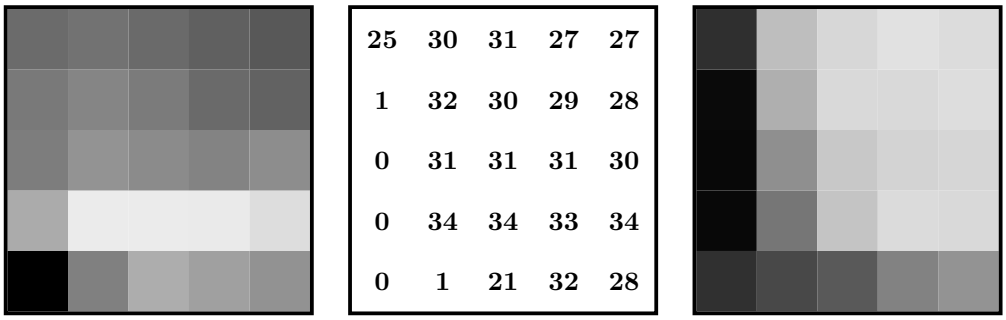
20	28	28	25	25
3	30	30	28	25
0	30	34	32	28
0	34	33	33	29
0	0	19	23	24

(a) Results for 0 left ants and 6 right ants.



35	39	37	36	35
4	39	38	37	37
0	40	39	38	37
0	39	39	39	38
0	5	35	37	36

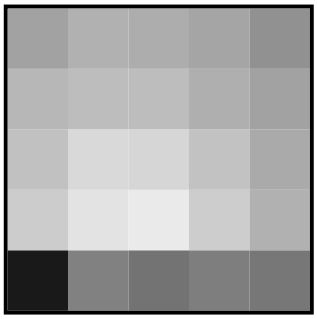
(b) Results for 3 left ants and 3 right ants.



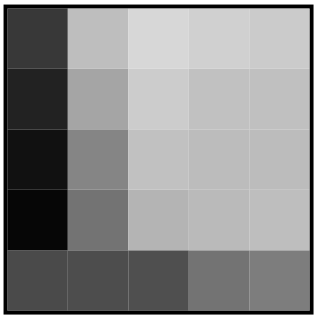
25	30	31	27	27
1	32	30	29	28
0	31	31	31	30
0	34	34	33	34
0	1	21	32	28

(c) Results for 6 left ants and 0 right ants.

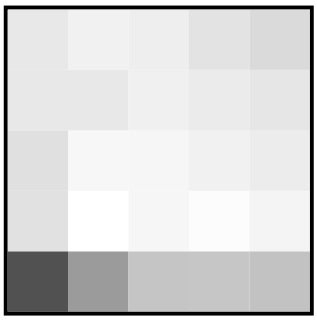
Figure 6.4: Results of the tuning of $\mathcal{MM}$AS concerning the instances with target sequence length $n = 409$. In each of the matrices the rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all" from top to bottom) and the columns correspond to the values of the determinism rate q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 296.748 corresponds to black color, and 400.018 to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 81.233 corresponds to black color, and 12.157 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.



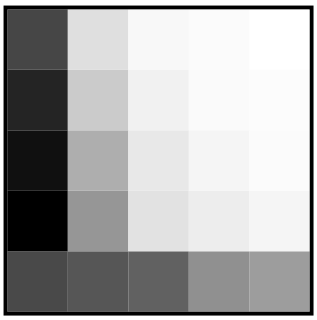
0	**26**	**27**	**23**	**20**
0	**27**	**27**	**25**	**22**
0	**27**	**32**	**27**	**22**
0	**28**	**32**	**27**	**23**
0	**0**	**11**	**15**	**14**



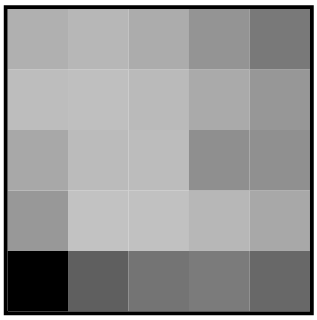
(a) Results for 0 left ants and 6 right ants.



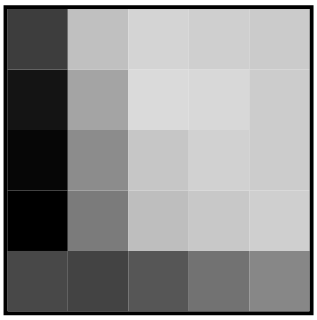
9	**35**	**36**	**35**	**34**
0	**35**	**36**	**36**	**35**
0	**36**	**36**	**36**	**36**
0	**37**	**38**	**38**	**37**
0	**0**	**26**	**30**	**27**



(b) Results for 3 left ants and 3 right ants.



3	**27**	**27**	**25**	**23**
0	**27**	**28**	**25**	**24**
0	**27**	**27**	**24**	**22**
0	**27**	**28**	**27**	**23**
0	**0**	**10**	**17**	**14**



(c) Results for 6 left ants and 0 right ants.

Figure 6.5: Results of the tuning of $\mathcal{MM}$AS concerning the instances with target sequence length $n = 509$. In each of the matrices the rows correspond to the 5 values of the size of the candidate list (2, 3, 5, 10 and "all" from top to bottom) and the columns correspond to the values of the determinism rate q (0.0, 0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 307.238 corresponds to black color, and 489.692 to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 170.43 corresponds to black color, and 31.757 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.

6.1.2 ACS tuning

In the tuning of ACS the parameters which have been tuned are *cls* (i.e., the size of the restricted candidate list), q (the determinism rate) and ξ (the rate for the reduction of pheromone). In this tuning n_f and n_b have been set to 3. We have decided to use this setting due to the results obtained during the $\mathcal{MM}$AS tuning presented in the previous section. Table 6.3 shows the possible values assigned to each considered parameter. Note that after the tuning of $\mathcal{MM}$AS we discarded some of the possible values of both *cls* and q.

Table 6.3: Tuning values for the ACS algorithm

Variable	value
cls	3, 5, 10
q	0.5, 0.75, 0.9, 0.95
ξ	0.01, 0.05, 0.1, 0.5

We have tested all the possible combinations of all the parameters; therefore 48 different configurations have been tested. Figure 6.6 show the results obtained by the application to the instances of length $l = 409$. We have only displayed this table because results in instances of different size l have produced similar results.

The first conclusion of the tuning is that, when comparing computation times, different values of *cls* and q produce the same effect than in the $\mathcal{MM}$AS algorithm: high values of *cls* combined with low values of q produce high computation times, whereas low values of *cls* and high values of q produce low computation times. In addition, low values of ξ result in lower computation times, where high values of ξ produce higher computation times. In general, the algorithm is slower than $\mathcal{MM}$AS .

Second, the high number of solved instances occurs with high values of determinism q and small values of *cls*. In general, low values of ξ also increase the number of solved instances. A notable result is that the configuration $q = 0.5$, $cls = 10$ and $\xi = 0.5$ does not solve any of the 200 instances.

Third, the similarity score values are, in general, quite similar in all the executions. For example, in instances of size 309, the difference between the best score and the lowest score is of approximately only 9 points (between 282.027 and 301.507). For this reason it is very difficult to identify which is the best configuration in terms of the similarity scores. As a general conclusion, due to its faster execution, and to its better results in terms of solved instances we propose to use the configuration $cls = 3$, $q = 0.95$ and $\xi = 0.01$, in addition to $\rho = 0.1$, $n_f = 3$ and $n_b = 3$, for the ACS algorithm.

6.1.3 ML-ACO tuning

The first parameter to be tuned in ML-ACO is the ACO algorithm that should be used in each level. To decide between the two algorithms described (that is, $\mathcal{MM}$AS and ACS) we present a graphic of the results of both ACO algorithms. For a further comparison between both algorithms we refer to the next section. Figure 6.7 shows both computation time and global similarity score for both algorithms. In this graphics we can see that the $\mathcal{MM}$AS algorithm is better than ACS, specially when the size of the instances increases. Therefore, we will use $\mathcal{MM}$AS to be applied in the multilevel framework.

Apart from the algorithm to be used, ML-ACO has some parameters to be tuned. These parameters are the time factor f_{time} which defines how much time is allocated to each level, and it_{wb} which is the number of iterations that an ACO algorithm may use at a certain level without finding a better solution (see Section 5.4.2). Table 6.4 shows the possible values assigned to each tuned parameter.

Table 6.4: Tuning values for the ML-ACO algorithm

Variable	value
f_{time}	1.25, 1.5, 2, 3, 5
it_{wb}	50, 100, 500

We have tested all the possible combinations of all the parameters; therefore 15 different configurations have been tested. Figures 6.8 and 6.9 show the results obtained in all the executions. We have drawn 3 graphics for each of the 5 instance types from the benchmark set by Błażewicz et al. [3]. The first one corresponds to the results concerning the global similarity score between the target sequence s_t and the obtained sequence (using the Needleman-Wunsch algorithm [29]). The second matrix corresponds to the number of solved instances (i.e. when $l(p)$ is optimal). The last one corresponds to the computation time. All the results have been calculated and displayed as described in Section 6.1.1.

From the results that are displayed in figures 6.8 and 6.9 we can draw the following conclusions. First, instances of size $n = 109$ are solved by all the configurations with the same results (all of them have been solved). Therefore we will not use those graphics to make further conclusions.

Second, the computation time of the algorithm increases when f_{time} decreases or when it_{wb} increases. Therefore longest computation times are found when f_{time} is low and and it_{wb} is high. However, in all the configurations the computation times are quite low.

Third, the results, concerning both the similarity scores and number of solved instances, are quite similar in all the configurations and it is quite difficult to make conclusions about the best parameters. Note that, for example, in instances of size $n = 509$, the difference between the best similarity score to the lowest is approximately of only 5 points. However, as a general result we may say that great values of it_{wb} combined with low values of f_{time} tend to generate lower number of solved instances.

Due to the similarity of the results the decision of choosing the best parameters is difficult. The only values that are really worse, in terms of computation time, are the ones obtained when $it_{wb} = 500$. Finally we have decided to use the parameters $f_{time} = 3$ and $it_{wb} = 100$ for the ML-ACO algorithm.

6.2 Results

In this section we will evaluate the results of the 3 ACO algorithms with the parameters chosen in the previous section. The detailed results of the executions can be found in Appendix A. In this section we will refer to these results.

Table 6.5: Results of $\mathcal{MM}$AS for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	80.00	159.84	239.97	319.89	399.74
Solved instances	40	40	40	40	40
Average similarity score (global)	108.40	201.63	297.89	397.84	487.01
Average similarity score (local)	108.70	205.61	303.92	399.79	494.53
Average computation time (sec)	0.014	1.68	5.52	16.14	41.57

6.2.1 $\mathcal{MM}$AS results

Tables A.1 to A.5 in Appendix A show the results of the executions of the $\mathcal{MM}$AS algorithm for the instances by Błażewicz et al. [3]. Additionally Table 6.5 shows the results in a compact form. In this table we have considered that a instance was solved if it was solved in at least one trial out of 10 trials.

From these results we may point out the following conclusions: In general, all results are very good. Most of the instances are solved, and obtain the target sequence s_t; from the 200 instances, 187 are solved in at least 9 out of 10 trials. Therefore we will focus our study in the ones that get bad results.

In most of the instances with bad results concerning the similarity score we have found some similarities. First of all, in most of these cases we can note high standard deviations. The reason for this may be that all the instances that cause bad results are characterized by the existence of at least one (sub-)optimal solution p_e which is very different from the desired solution p_c (the one which resembles the target sequence s_t). If, in the first stages of the ACO algorithm, a path similar to p_e—or p_e itself— is found, the ACO algorithm will tend to improve this path. As it is very different from the real path p_c it would be difficult to find p_c due to the increase of the pheromone values on the components of p_e. A particular case of this situation occurs when p_e solves the SBH problem (i.e., $l(p_e) = l(p_c)$). In this case the algorithm will never find the target sequence because it stops. These situations occur for example in instances 34 of size 209, 28 of size 409 or in both instances 16 and 39 of size 509.

Another notable peculiarity is found in instance 33 of size 309. This instance is solved in all 10 trials. However, the obtained similarity scores are low. Despite bad results in the global similarity score, the results concerning the local similarity score are not that bad (49.8 and 216.3 respectively). This result is a consequence of the deficiency of the model described in Section 2.2.1. This error appears when a sequence of type $s_e = s_1 + s_0$ is created, where the target sequence is $s_t = s_0 + s_1$. Solution s_e compared with s_t with a global alignment method results in a bad score, but in local alignment it may be quite good. This result is also shown in instance 34 of size 209.

6.2.2 ACS results

Tables A.6 to A.10 in Appendix A show the results of the executions of the ACS algorithm for the instances by Błażewicz et al. [3]. Additionally Table 6.6 shows the results in a compact form. In this table we have considered that a instance was solved if it was solved in at least one out of 10 trials.

Table 6.6: Results of ACS for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	79.97	159.54	239.47	318.67	398.56
Solved instances	40	39	40	39	36
Average similarity score (global)	108.40	198.41	297.3	368.62	444.16
Average similarity score (local)	108.70	204.21	301.01	380.73	465.47
Average computation time (sec)	0.05	1.15	10.86	34.39	81.78

Table 6.7: Results of ML-ACO for the instances by Błażewicz et al. [3].

Spectrum size	100	200	300	400	500
Average solution quality	80.00	159.91	240.00	319.78	399.86
Solved instances	40	40	40	40	39
Average similarity score (global)	108.40	206.67	294.24	394.55	499.02
Average similarity score (local)	108.70	207.66	300.88	396.87	503.6
Average computation time (sec)	0.042	0.36	0.31	3.5	7.9

From the results that are provided in these tables we can draw the following conclusions. First, the results are, in all aspects, worse than the results of the $\mathcal{MM}$AS algorithm. Second, the standard deviations of all the measures are high, because in general, different executions of the algorithm for the same instance get different results. This shows that the ACS algorithm is not very robust in contrast to the $\mathcal{MM}$AS algorithm; the results of the algorithm depend a lot on the first solutions found. It is also notable that when the instance size increases the number of solved instances decreases.

Finally, with respect to the instances that $\mathcal{MM}$AS had difficulties to solve, we do not see any improvement by ACS.

6.2.3 ML-ACO results

Tables A.11 to A.15 in Appendix A show the results of the applications of the ML-ACO algorithm to the instances by Błażewicz et al. [3]. Additionally Table 6.7 shows the results in a compact form. In this table we have considered that a instance was solved if it was solved in at least one out of 10 trials.

From the results that are displayed in these tables we can draw the following conclusions. First, the results are, in all aspects, very good. Nearly all the instances are solved, and many target sequence s_t are re-constructed; from the 200 instances, 194 are solved in at least 9 out of 10 trials, and 193 are solved in all the trials.

Second, the standard deviations of all the measures are, in general, very low. This means that different executions of the algorithm for the same instance get the same (or very similar) results; therefore, ML-ACO algorithm is very robust. Notice, that the instances are either solved in all the trials, or nearly in none.

Finally, concerning the instances that $\mathcal{MM}$AS had difficulties to solve, we see an improvement in some of them. However, most of the instance that were not solved by $\mathcal{MM}$AS

are not solved by ML-ACO neither.

In order to study better the multilevel framework, we have drawn 5 graphics which represent in which level the best solution of a run was found. We have displayed the results for the 400 executions of each size (10 trials for each of the 40 instances). In addition we have added to the graphic the number of the solutions that solved the model, represented by a dashed line. When all the best solutions are solved solutions, no dashed line is drawn.

From the results that are displayed in these figures we can draw the following conclusions. First, the multilevel strategy is quite effective, because most of the solutions are found in levels different from the original instance G^0; only about a 5% of the solutions are found in the original instance. This explains the big advantage of ML-ACO over $\mathcal{MM}$AS in terms of computation time.

Second, when instances grow, the results tend to be found in smaller levels. For example, in instances of size $n = 209$ about 70 executions where solved in the smallest step, whereas in the instances of size $n = 509$ more than 200 where solved in the smallest level.

6.2.4 Comparison

In order to compare between the three ACO algorithms namely, $\mathcal{MM}$AS, ACS, and ML-ACO, we have drawn 2 graphs in Figure 6.12 which show a comparison both in terms of the global similarity score and the computational time of the algorithms. From these graphs we can draw the following conclusions. First, in terms of score, both $\mathcal{MM}$AS and ML-ACO are better than ACS, specially in big instances. $\mathcal{MM}$AS and ML-ACO get, in general the same results except for the instances with $n = 509$ where ML-ACO seems to be better. Furthermore, when we look at the results of appendix A, we can see that the standard deviations of the score are lower in ML-ACO than in $\mathcal{MM}$AS . Therefore, ML-ACO is more robust than $\mathcal{MM}$AS.

Second, the $\mathcal{MM}$AS algorithm is faster than ACS. Furthermore, the application of the multilevel strategy decreases quite a lot the execution time of the algorithm, and ML-ACO is the fastest of the three algorithms.

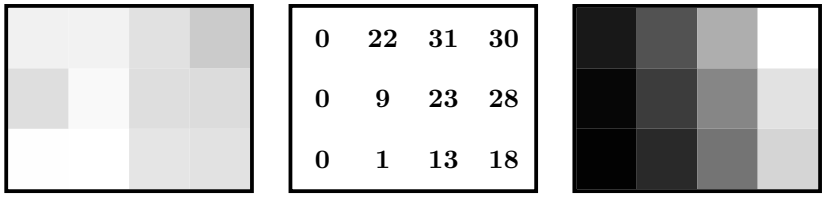
Finally, in Figure 6.13 we present a comparison between the three ACO algorithms, the constructive heuristic HSM described in Section 3.6 (which was our best construction heuristic) and the best available meta-heuristic approaches from the literature, namely, the evolutionary algorithms EA1 to EA3, the tabu search TS, and the hybrid tabu search with scatter search TS/SS, all of them are cited in Section 2.3.2. In this graph we can see that the three ACO algorithms are better than all the other heuristics except for EA2. EA2 is the only heuristic which can be compared to ML-ACO, and in fact, gets very similar results.

6.3 Final tests

In order to evaluate more precisely the algorithms we have applied the two best ACO algorithms ($\mathcal{MM}$AS and ML-ACO) and the constructive heuristic HSM to some of the sequencing instances proposed in [24]. This benchmark set consists of 100 random DNA sequences of size 100, 200,..., and 600. Furthermore 3 different spectra exist for each target sequence. The difference lies in the length of the oligonucleotides. More in detail, the 3 different spectra were generated with $l = 8$, $l = 9$ and $l = 10$. The spectra contain 20% of false negative errors and 20% of false positive errors (randomly generated). Therefore there are 100 instances of each size, with 3 different spectrum sizes, which gives a total of 3600 instances. In this section we present the results of these tests.

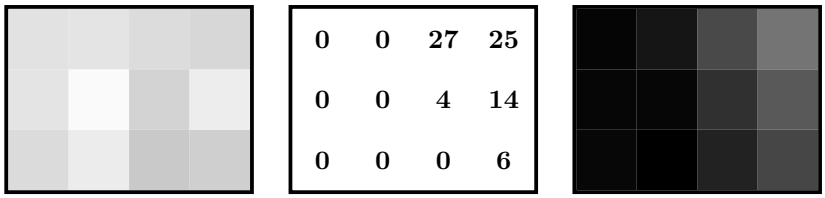
Figure 6.14 contains 3 graphs, each one containing the global similarity score results for one value of l. In these graphs, the score value is given in percent (i.e., the score value has been divided by twice the size of the target sequence). In Figure 6.15 we have drawn 3 graphs, one per value of l, containing the computation times of the algorithms.

From the results that are displayed in these figures we can draw the following conclusions. First, in all the algorithms when l decreases the global similarity score also decreases and the computation time needed increases. Therefore instances of smaller l are more difficult to solve than instances of greater l. Second, for all sizes of l, ML-ACO is the best algorithm in terms of score followed by $\mathcal{MM}$AS . Finally, the computation time of $\mathcal{MM}$AS increases when l increases, but this result is specially remarkable in ML-ACO. When probes are of size $l = 10$ ML-ACO is much faster than $\mathcal{MM}$AS. However, in instances of $l = 8$ ML-ACO is comparable to $\mathcal{MM}$AS in computation time, and in some cases is even slower, specially in instances of big size. From another point of view, the computation time of ML-ACO strongly decreases when l is increased. The reason of the high computation time of ML-ACO in small sizes of l is that, when l is small, the contraction method easily introduces errors in the generated instances, and, it is impossible to solve the model in this level. Therefore, ML-ACO wastes computation time by being applied to levels in which the problem cannot be solved.



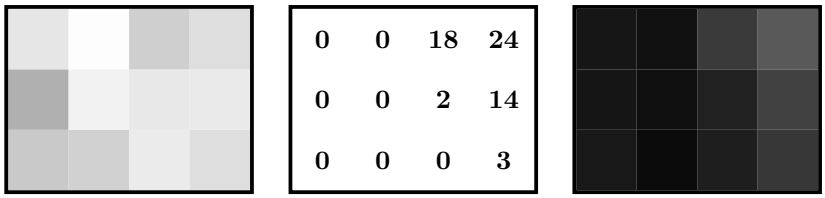
0	**22**	**31**	**30**
0	**9**	**23**	**28**
0	**1**	**13**	**18**

(a) Results for $\xi = 0.01$.



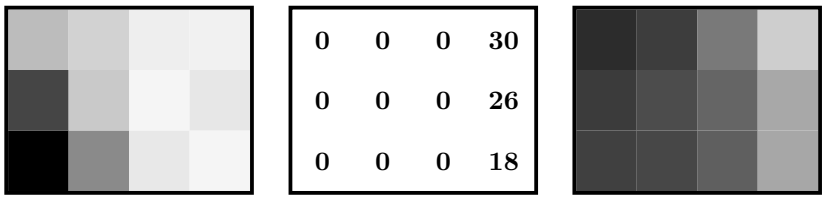
0	**0**	**27**	**25**
0	**0**	**4**	**14**
0	**0**	**0**	**6**

(b) Results for $\xi = 0.05$.



0	**0**	**18**	**24**
0	**0**	**2**	**14**
0	**0**	**0**	**3**

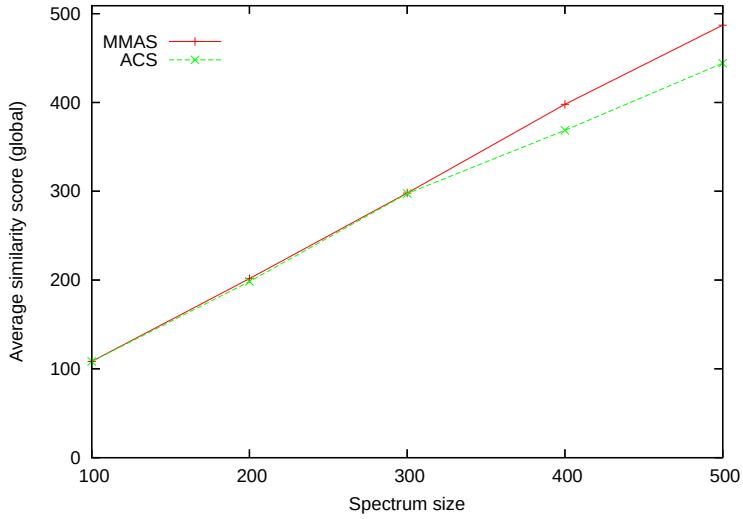
(c) Results for $\xi = 0.1$.



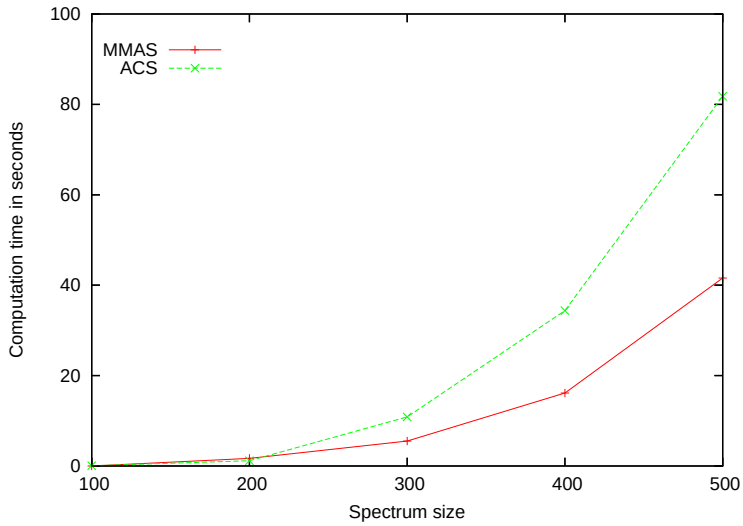
0	**0**	**0**	**30**
0	**0**	**0**	**26**
0	**0**	**0**	**18**

(d) Results for $\xi = 0.5$.

Figure 6.6: Results of the tuning of ACS concerning the instances with target sequence length $n = 409$. In each of the matices the rows correspond to the 3 values of the size of the candidate list (3, 5 and 10 from top to bottom) and the columns correspond to the values of the determinism rate q (0.5, 0.75, 0.9 and 0.95 from left to right). The first matrix of each sub-figure shows the values of the similarity score (where 313.302 corresponds to black color, and 382.795 corresponds to white color) the second matrix shows the number of solved instances, and the last one shows the computation time (where 90.643 corresponds to black color, and 34.294 corresponds to white color). Note that in the first and last matrix of each sub-figure, light gray represents the best solutions and dark gray the worst.

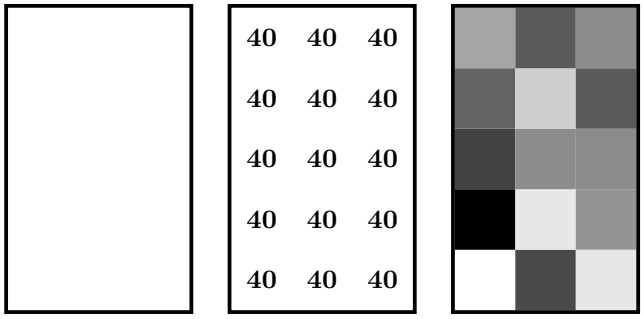


(a) Global average similarity score



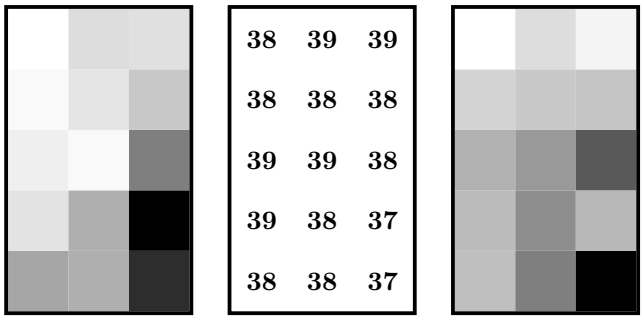
(b) Computation time

Figure 6.7: Comparison between the ACS and the $\mathcal{MM}$AS algorithms. The comparison concerns the instances of Błażewicz et al. [3].



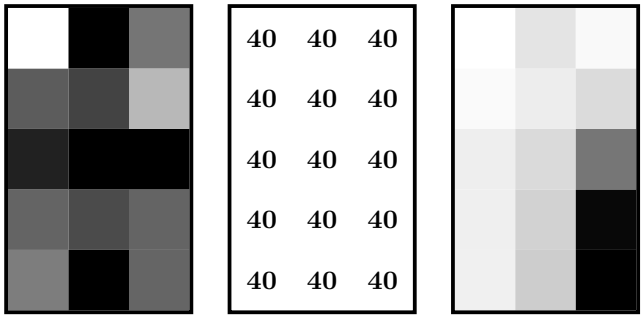
40	40	40
40	40	40
40	40	40
40	40	40
40	40	40

(a) Results for instances with target sequence length 109. In the score matrix 108.4 corresponds to black color, and 108.4 to white color. In the time matrix 0.005 corresponds to black color, and 0.005 to white color.



38	39	39
38	38	38
39	39	38
39	38	37
38	38	37

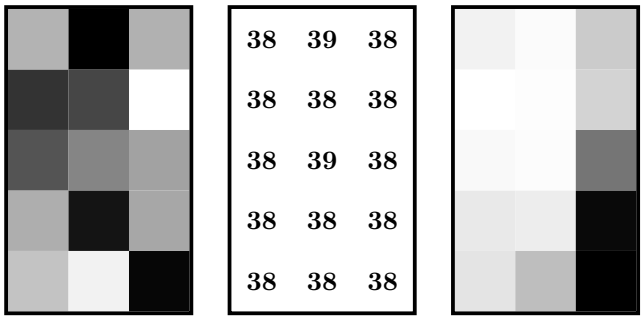
(b) Results for instances with target sequence length 209. In the score matrix 203.363 corresponds to black color, and 207.065 to white color. In the time matrix 0.669 corresponds to black color, and 0.267 to white color.



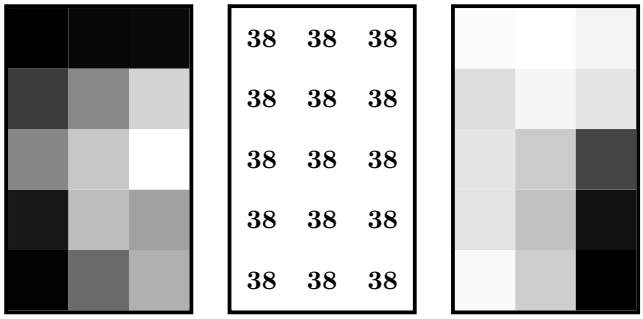
40	40	40
40	40	40
40	40	40
40	40	40
40	40	40

(c) Results for instances with target sequence length 309. In the score matrix 292.925 corresponds to black color, and 297.877 to white color. In the time matrix 1.459 corresponds to black color, and 0.22 to white color.

Figure 6.8: Results of the tuning for instances with target sequence sizes (a) $n = 109$, (b) $n = 209$ and (c) $n = 309$. In each of the matrices the rows correspond to the 5 values of f_{time} (1.25, 1.5, 2, 3 and 5 from bottom to top) and the columns correspond to the values of it_{wb} (50, 100, 500 from left to right). The first matrix of each sub-figure shows global similarity score values, the second matrix shows the number of solved instances, and the last one shows the computation times. Note that in the first and last matrix, light gray represents the best solutions and dark gray the worst.

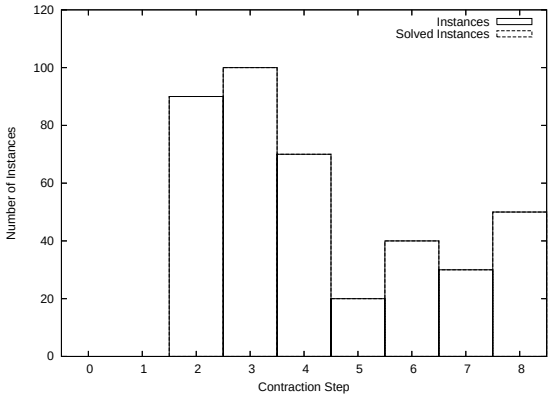


(a) Results for instances with target sequence length 409. In the score matrix 392.988 corresponds to black color, and 398.663 to white color. In the time matrix 8.372 corresponds to black color, and 3.447 to white color.

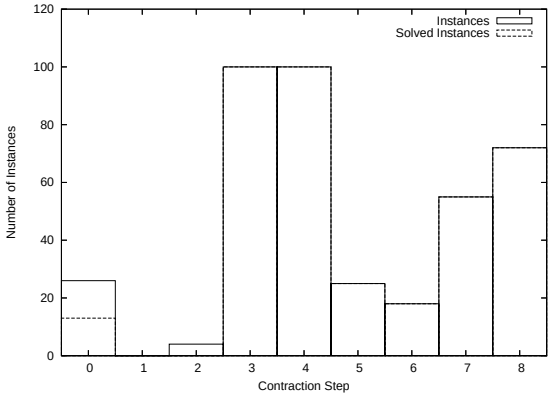


(b) Results for instances with target sequence length 509. In the score matrix 496.52 corresponds to black color, and 501.207 to white color. In the time matrix 19.668 corresponds to black color, and 7.385 to white color.

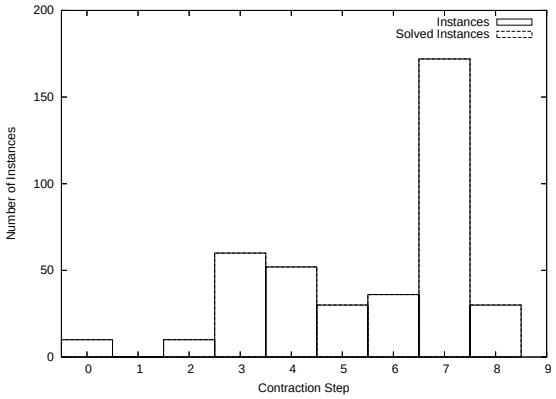
Figure 6.9: Results of the tuning for instances with target sequence sizes (a) $n = 409$ and (b) $n = 509$. In each of the matrices the rows correspond to the 5 values of f_{time} (1.25, 1.5, 2, 3 and 5 from bottom to top) and the columns correspond to the values of it_{wb} (50, 100, 500 from left to right). The first matrix of each sub-figure shows global similarity score values, the second matrix shows the number of solved instances, and the last one shows the computation times. Note that in the first and last matrix, light gray represents the best solutions and dark gray the worst.



(a) Number of best and optimal solutions found in each step for instances of size $n = 100$

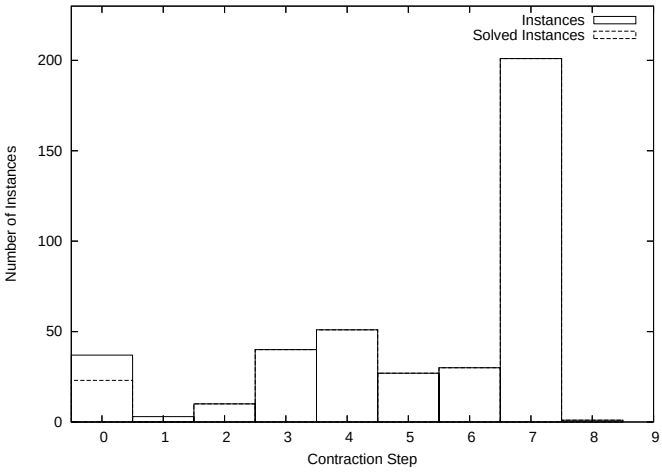


(b) Number of best and optimal solutions found in each step for instances of size $n = 200$

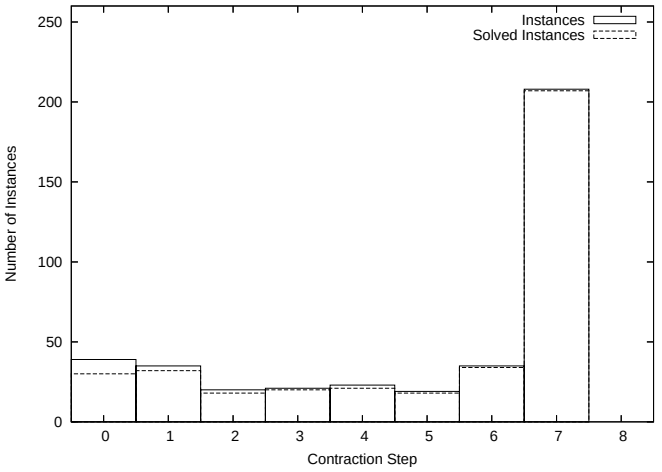


(c) Number of best and optimal solutions found in each step for instances of size $n = 300$

Figure 6.10: Number of best and optimal solutions found in each level of the multilevel framework

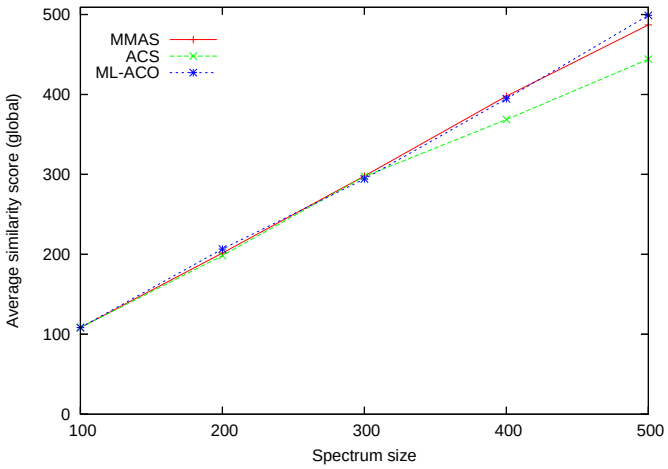


(a) Number of best and optimal solutions found in each step for instances of size $n = 400$

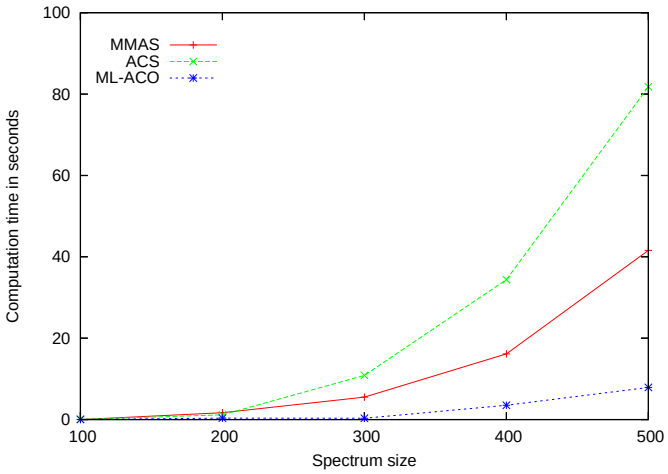


(b) Number of best and optimal solutions found in each step for instances of size $n = 500$

Figure 6.11: Number of best and optimal solutions found in each level of the multilevel framework



(a) Comparison between $\mathcal{MM}$AS , ACS and ML-ACO concerning the global similarity scores



(b) Comparison between $\mathcal{MM}$AS , ACS and ML-ACO concerning computation times

Figure 6.12: Comparison between the $\mathcal{MM}$AS , ACS and ML-ACO algorithms

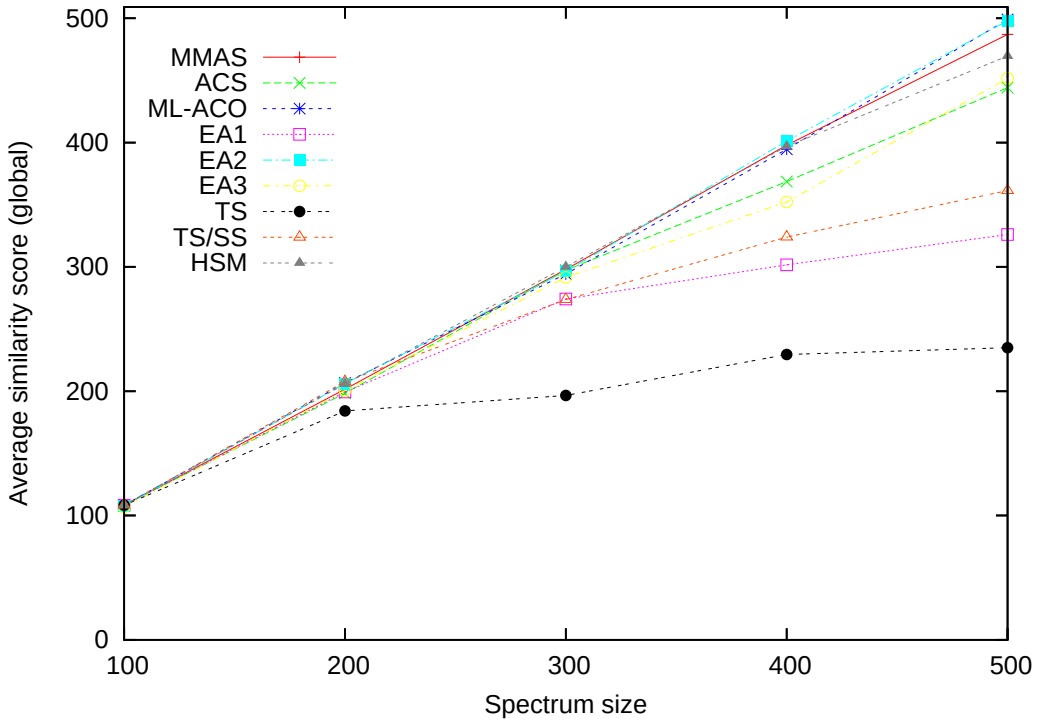
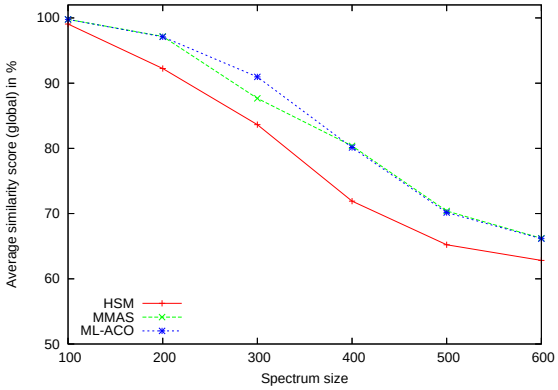
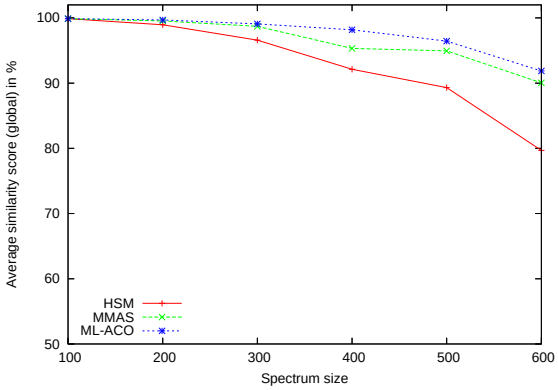


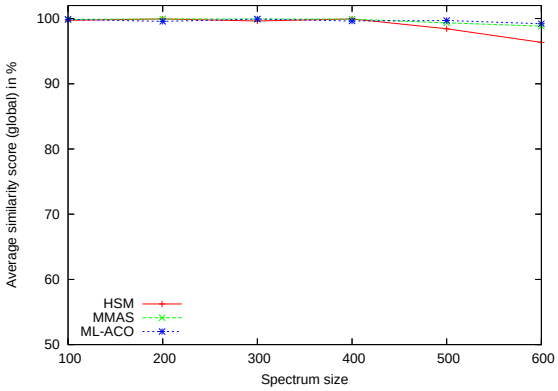
Figure 6.13: Comparison between all the existing (meta-)heuristics for the SBH problem concerning the global similarity scores



(a) Global similarity scores of HSM, $\mathcal{MM}$AS and ML-ACO of instances of oligonucleotide length $l = 8$

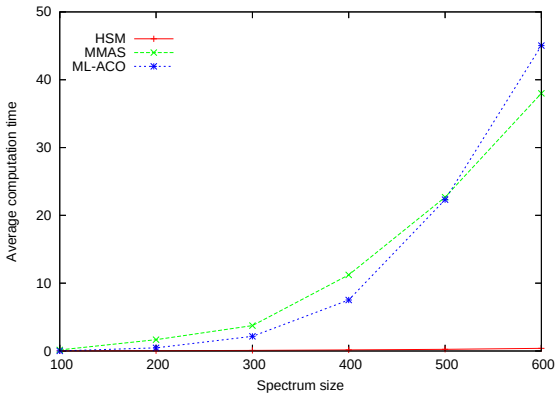


(b) Global similarity scores of HSM, $\mathcal{MM}$AS and ML-ACO of instances of oligonucleotide length $l = 9$

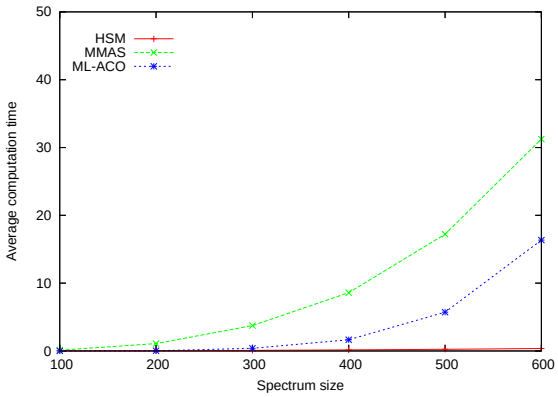


(c) Global similarity scores of HSM, $\mathcal{MM}$AS and ML-ACO of instances of oligonucleotide length $l = 10$

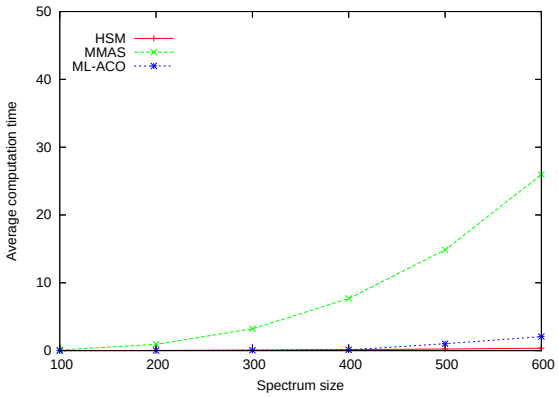
Figure 6.14: Comparison between the HSM, $\mathcal{MM}$AS and ML-ACO algorithms concerning the global similarity score.



(a) Computation time of HSM, $\mathcal{MM}$AS and ML-ACO for instances with oligonucleotide length $l = 8$



(b) Computation time of HSM, $\mathcal{MM}$AS and ML-ACO for instances with oligonucleotide length $l = 9$



(c) Computation time of HSM, $\mathcal{MM}$AS and ML-ACO for instances with oligonucleotide length $l = 10$

Figure 6.15: Comparison between HSM, $\mathcal{MM}$AS and ML-ACO concerning the computation times.

Chapter 7

Summary

In this last chapter we present an overview of the contents of this thesis. We will start by defining the problem. We will cite some constructive heuristics and comment on their results. Finally we will shortly describe the Ant Colony Optimization meta-heuristic and its application to the problem.

7.1 The problem

In this thesis we studied the Sequencing by Hybridization problem (SBH). The biological problem consists in determining the exact structure of a DNA molecule, called DNA sequencing. Sequencing by Hybridization is a method to sequence DNA. It works roughly as follows: the first phase of the method consists of a chemical experiment which requires a so-called DNA array. A DNA array is a two-dimensional grid whose cells typically contain all possible DNA strands—called probes—of equal length l. After the generation of the DNA array, the chemical experiment is started. It consists of bringing together the DNA array with many copies of the DNA sequence to be read, also called the DNA target sequence. Hereby, the target sequence might react with a probe on the DNA array if and only if the probe is a subsequence of the target sequence. After the experiment, the DNA array allows the identification of the probes that reacted with target sequences. This subset of probes is called the spectrum. The second phase of the sequencing by hybridization technique consists in using the spectrum to determine the unknown DNA target sequence. The reconstruction of the original sequence consists of finding an order of the spectrum elements in which each pair of neighboring elements overlaps on $l-1$ letters (i.e., the last $l-1$ letters of each probe coincide with the first $l-1$ letters of the next). However, the hybridization experiment usually produces errors in the spectrum. There are two types of errors:

1. **Negative errors:** Some probes that should be in the spectrum (because they appear in the target sequence) do not appear in the spectrum.

2. **Positive errors:** A probe of the spectrum that does not appear in the target sequence is called a positive error.

The computational part of the hybridization experiment is to achieve the reconstruction of the target sequence with the oligonucleotides obtained in the first phase (i.e., the spectrum). The existence of errors in the spectrum results in strongly NP-hard combinatorial problems, as shown by Błażewicz and Kasprzak in [7].

In order to tackle the problem we focused on the optimization problem that was introduced as a model for DNA sequencing by hybridization by Błażewicz et al. in [3]. The model is roughly as follows: the input of the problem is a set S (spectrum) of words of equal length l over the alphabet {A,C,G,T} (i.e., $S = \{\{A, C, G, T\}^l\}^*$) and the length n of the original DNA target sequence. The goal is to find a sequence of length $\leq n$ containing the maximal number of elements of S.

In Section 2.2.2, we presented how this model can be studied as a graph problem. Mainly, each oligonucleotide is represented by a node in a fully connected directed graph. Each edge $e_{s,s'}$ of the graph has a weight whose value is the size of the largest sequence that is both a suffix of s and a prefix of s'; that is the overlap between the two sequences. The objective of SBH is to find a directed path p in the graph of maximum length such that the length of the obtained DNA sequence is less or equal to the size of the target sequence.

Finally, in Section 2.3, we introduced the Combinatorial Optimization problems. In general many optimization problems of practical as well as theoretical importance consist of the search for a "best" configuration of a set of variables to achieve some goals. They seem to divide naturally into two categories: those where solutions are encoded with *real-valued* variables, and those where solutions are encoded with *discrete* variables. Among the later ones we find a class of problems called Combinatorial Optimization (CO) problems. According to Papadimitriou and Steigliz [30], in CO problems, we are looking for an object from a finite—or possibly countably infinite—set. This object is typically an integer number, a subset, a permutation or a graph structure. For CO problems that are NP-hard, no polynomial time algorithm exists, assuming that P $\neq$ NP. Therefore, complete methods might need exponential time in the worst-case. Thus, the use of approximate methods to solve CO problems has received more and more attention in the last 30 years. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Among basic approximate methods we usually distinguish between constructive methods and local search methods. Constructive algorithms generate solutions from scratch by adding—to an initially empty partial solution—components, until a solution is complete. Local search algorithms start from some initial solution and iteratively try to replace the current solution by a better solution in an appropriately defined neighborhood of the current solution. In the last 20 years, a new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called meta-heuristics. This class of algorithms includes Ant Colony Optimization (CO), Evolutionary Computation (EC) including Genetic Algorithms (GA), the Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Simulated Annealing (SA), and Tabu Search (TS).

7.2 First attempts to tackle the problem: constructive heuristics

In Chapter 3 we presented some constructive heuristics to tackle SBH. We first presented 4 heuristics, namely LAG, SH, FB-LAG, FB-SH, based on a Look-Ahead Greedy that is proposed in [3]. The general idea of these heuristics is to start the path construction in graph G with one of the probes of the spectrum, and to extend this path in a step-by-step manner. Afterwards we presented a heuristic, namely SM. The idea of this heuristic is conceptionally quite different

to the first ones. Instead of constructing one single path, the SM heuristic starts with a set P of $|S|$ paths, each of which only contains exactly one oligonucleotide $s \in S$, and then merges paths until a path of sufficient size is obtained. Finally we presented two hybrids, namely HSM and S-HSM, of SM and FB-LAG, and SM and FB-SH respectively.

In order to test these heuristics we applied them to a set of benchmark instances for DNA sequencing by hybridization introduced by Błażewicz et al. in [3].

The results were really good. In general terms, variations over the original LAG improved the initial algorithm. Moreover the results of SM were clearly better than the results of the lineal construction heuristics (LAG , FB-LAG , SH and FB-SH). The best results were obtained by the hybrid heuristics (both HSM and S-HSM). When comparing between them we saw that HSM improved over S-HSM . Even for the largest problem instances, the HSM heuristic produced sequences with very high similarity scores. Finally we compared the results of HSM with the best available meta-heuristic approaches from the literature. The results where surprising: HSM was clearly better than most meta-heuristic approaches. Furthermore, the results of HSM where—except for the problem instance of greater target sequence size—comparable to the results of the best meta-heuristic approach.

7.3 Ant Colony Optimization: ACO

In order to create more powerful algorithms to tackle the SBH problem, we used the Ant Colony Optimization meta-heuristic described in Chapter 4. Ant Colony Optimization is a meta-heuristic approach proposed in [17, 20, 18]. The inspiring source of ACO is the foraging behavior of real ants. This behavior (as described by Deneubourg et al. in [16]) enables ants to find shortest paths between food sources and their nest. While walking from food sources to the nest and vice versa, ants deposit a substance called pheromone on the ground. When they decide about a direction to go, they choose with higher probability paths that are marked by stronger pheromone concentrations. This basic behavior is the basis for a cooperative interaction which leads to the emergence of shortest paths.

ACO algorithms are based on a parametrized probabilistic model–the pheromone model–that is used to model the chemical pheromone trails. Artificial ants incrementally construct solutions by adding opportunely defined solution components to a partial solution under consideration. For doing that, artificial ants perform randomized walks on a graph whose vertexes are the solution components and edges are the connections. Components and connections can have associated a pheromone trail parameter. Furthermore, components and connectors can have associated a heuristic value representing a priori or run time heuristic information about the problem instance. Both pheromone and heuristic values are used by ants to make probabilistic decisions on how to move on the construction graph; ants will chose with higher probability connections and components whose pheromone and heuristic values are high. Moreover, the pheromone values are generally updated by the following rule: components and connections used by solutions of high quality (according to the problem) increase their pheromone value.

Many variations on the original ACO meta-heuristic exist. In Chapter 4 we explained the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS), the Ant Colony System (ACS), the Hyper Cube Framework (HCF), and the Multi-level ACO (ML-ACO).

7.4 ACO algorithms for SBH

Finally, in Chapter 5 we proposed two different adaptations of ACO algorithms to solve the SBH problem. First a $\mathcal{MAX}$-$\mathcal{MIN}$ ant system ($\mathcal{MM}$AS) implemented in the hyper-cube framework (HCF) was described. Using this first implementation as a basis, we proposed some changes of the algorithm that produced an Ant Colony System (ACS) also implemented in the HCF. Finally we introduced a multi-level framework for the problem based on the SM heuristic (ML-ACO). Furthermore, in Chapter 6, we applied the algorithms to a set of benchmark instances for DNA sequencing by hybridization introduced by Błażewicz et al. in [3] to tune the parameters of these algorithms. When we compared the two pure ACO algorithms, we saw that $\mathcal{MM}$AS was better than ACS. Therefore we applied the $\mathcal{MM}$AS algorithm within the multi-level framework. The results of this final algorithm were very good. The computation times of ML-ACO were much lower than the $\mathcal{MM}$AS computation times. Furthermore, the results of ML-ACO also improved over the results of $\mathcal{MM}$AS .

When we compared the results of the three algorithms with the best available meta-heuristic approaches from the literature we saw that the three ACO algorithms were better than all the others except for one, namely EA2 [23]. EA2 was the only heuristic which could be compared to ML-ACO, and in fact, got very similar results.

Finally, we applied HSM, $\mathcal{MM}$AS , and ML-ACO to a set of benchmark instances for DNA sequencing by hybridization introduced by E. R. Fernandes and C. C. Ribeiro in [24]. With these instances we confirmed that ML-ACO was the best of the three heuristics. Finally we saw that, when the size of the oligonucleotides increases, the improvement of ML-ACO over $\mathcal{MM}$AS also increased.

7.5 Accomplished objectives

To finish this thesis we want to remark the objectives that have been accomplished:

- We have proposed high quality constructive algorithms for the SBH problem.
- We have proposed two ACO algorithms. One of them with good results.
- We have proposed a multi-level approach to the SBH and we applied our best ACO algorithm in this framework obtaining an algorithm comparable to the best method described in the current literature.

Appendix A

Results of the algorithms

This appendix contains detailed results of the ACO algorithms developed in this thesis.

A.1 $\mathcal{MM}$AS results

Tables A.1 to A.5 show the results of the application of $\mathcal{MM}$AS to all the instances by Błażewicz et al. [3]. The results are given as averages over 10 trials and the standard desviation in parenthesis. The first column is the number of the instance. The second one—*solved*—is the number of solved instances in 10 trials (i.e, its a value from 0 to 10). The third column—*quality*— provides the average solution quality. The forth and fifth columns—*global* and *local*—provide both average global and local similarity scores (using Needleman-Wunsch algorithm, and Smith-Waterman algorithm respectively). Finally, the sixth column—*time*—provides the average computation time for solving an instance (in seconds).

A.2 ACS results

Tables A.6 to A.10 show the results of the application of ACS to all the instances by Błażewicz et al. [3]. The results are given as averages over 10 trials and the standard desviation in parenthesis. The first column is the number of the instance. The second one—*solved*—is the number of solved instances in 10 trials (i.e, its a value from 0 to 10). The third column—*quality*— provides the average solution quality. The forth and fifth columns—*global* and *local*—provide both average global and local similarity scores (using Needleman-Wunsch algorithm, and Smith-Waterman algorithm respectively). Finally, the sixth column—*time*—provides the average computation time for solving an instance (in seconds).

A.3 ML-ACO results

Tables A.11 to A.15 show the results of the application of ML-ACO to all the instances by Błażewicz et al. [3]. The results are given as averages over 10 trials and the standard desviation in parenthesis. The first column is the number of the instance. The second one—*solved*—is the number of solved instances in 10 trials (i.e, its a value from 0 to 10). The third column—*quality*— provides the average solution quality. The forth and fifth columns—*global* and *local*—provide both average global and local similarity scores (using Needleman-Wunsch

Table A.1: Results of $\mathcal{MM}$AS for the instances of size 109.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.11	(0.06)
2	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.15	(0.09)
3	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.12	(0.05)
4	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.11	(0.05)
5	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.19	(0.12)
6	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.1	(0.05)
7	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.4	(0.36)
8	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.1	(0.04)
9	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.11	(0.06)
10	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.12	(0.06)
11	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.17	(0.06)
12	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.08	(0.03)
13	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.12	(0.06)
14	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.27	(0.15)
15	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.14	(0.05)
16	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.08	(0.05)
17	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.08	(0.05)
18	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.16	(0.07)
19	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.14	(0.09)
20	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.11	(0.05)
21	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.12	(0.06)
22	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.15	(0.07)
23	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.1	(0.05)
24	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.19	(0.04)
25	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.14	(0.05)
26	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.13	(0.08)
27	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.13	(0.06)
28	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.14	(0.04)
29	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.15	(0.08)
30	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.09	(0.08)
31	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.12	(0.04)
32	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.14	(0.06)
33	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.11	(0.05)
34	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.13	(0.05)
35	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.14	(0.07)
36	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.15	(0.06)
37	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.13	(0.05)
38	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.09	(0.06)
39	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.11	(0.07)
40	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.12	(0.06)
Summary	399	80.0	(0.05)	108.4	(1.11)	108.7	(0.56)	0.14	(0.1)

Table A.2: Results of $\mathcal{MM}$AS for the instances of size 209.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	1.16	(0.25)
2	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.11	(0.28)
3	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.1	(0.38)
4	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.07	(0.2)
5	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.05	(0.41)
6	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	1.09	(0.26)
7	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	2.1	(2.1)
8	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.56	(0.32)
9	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.72	(0.22)
10	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	1.63	(0.27)
11	8	159.0	(2.83)	200.3	(27.51)	203.7	(16.76)	3.94	(2.98)
12	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.16	(0.29)
13	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.04	(0.26)
14	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.47	(0.25)
15	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.19	(0.32)
16	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.22	(0.24)
17	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.11	(0.24)
18	8	159.8	(0.42)	209.0	(0.0)	209.0	(0.0)	4.33	(3.38)
19	4	158.8	(1.14)	175.1	(41.23)	194.5	(17.45)	3.25	(1.98)
20	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	1.36	(0.41)
21	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.02	(0.25)
22	10	160.0	(0.0)	200.7	(4.35)	205.9	(1.45)	1.24	(0.2)
23	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.4	(0.51)
24	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	2.46	(2.0)
25	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	1.02	(0.45)
26	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.31	(0.25)
27	9	159.6	(1.26)	207.4	(5.06)	207.8	(3.79)	4.43	(2.75)
28	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	1.88	(1.49)
29	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.21	(0.24)
30	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	2.12	(2.3)
31	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.03	(0.46)
32	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	1.34	(0.24)
33	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.96	(0.21)
34	1	158.2	(0.63)	20.0	(66.41)	123.5	(30.04)	2.4	(2.13)
35	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.15	(0.27)
36	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	1.14	(0.21)
37	2	158.1	(1.1)	177.9	(48.24)	194.0	(15.52)	3.52	(2.01)
38	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.83	(0.36)
39	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	2.16	(2.05)
40	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.08	(0.29)
Summary	371	159.84	(0.71)	201.63	(33.28)	205.61	(14.95)	1.68	(1.52)

Table A.3: Results of $\mathcal{MM}$AS for the instances of size 309.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.32	(0.53)
2	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.75	(0.42)
3	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.36	(0.73)
4	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.29	(1.01)
5	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.77	(3.64)
6	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.04	(0.7)
7	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.75	(3.29)
8	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.7	(0.78)
9	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.74	(1.12)
10	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.18	(0.96)
11	9	239.4	(1.9)	281.7	(86.33)	293.0	(50.6)	27.47	(18.3)
12	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.08	(0.55)
13	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.42	(0.85)
14	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	5.04	(3.1)
15	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.5	(0.88)
16	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.59	(0.8)
17	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.51	(2.91)
18	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.22	(0.9)
19	10	240.0	(0.0)	250.8	(93.71)	254.7	(87.43)	4.78	(3.27)
20	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.6	(0.84)
21	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.35	(0.67)
22	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.01	(0.51)
23	10	240.0	(0.0)	303.0	(0.0)	306.0	(0.0)	3.46	(0.68)
24	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.99	(0.79)
25	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	14.42	(11.09)
26	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.34	(0.75)
27	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.78	(3.59)
28	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	8.8	(5.64)
29	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.28	(0.78)
30	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.84	(0.64)
31	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	4.13	(0.85)
32	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	5.13	(3.24)
33	10	240.0	(0.0)	49.8	(91.07)	216.3	(32.57)	4.55	(1.45)
34	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	8.81	(7.3)
35	10	240.0	(0.0)	239.5	(69.04)	285.5	(22.66)	4.47	(1.13)
36	6	239.6	(0.52)	305.4	(2.07)	306.4	(2.07)	14.72	(15.36)
37	10	240.0	(0.0)	305.0	(0.0)	307.0	(0.0)	3.9	(0.63)
38	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	3.79	(0.88)
39	7	239.6	(0.7)	306.3	(8.54)	306.8	(6.96)	11.06	(7.78)
40	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	3.74	(0.69)
Summary	392	239.97	(0.34)	297.89	(49.49)	303.92	(23.48)	5.52	(6.39)

Table A.4: Results of $\mathcal{MM}$AS for the instances of size 409.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	14.91	(8.26)
2	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	9.36	(5.77)
3	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	6.81	(1.1)
4	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	17.6	(14.76)
5	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	8.06	(2.13)
6	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	25.54	(23.09)
7	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	13.34	(12.79)
8	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	7.87	(1.0)
9	6	318.0	(3.02)	357.8	(82.44)	373.0	(58.02)	39.84	(28.23)
10	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	9.08	(1.24)
11	8	319.6	(0.84)	354.0	(115.95)	359.0	(105.41)	42.76	(24.46)
12	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	7.85	(0.71)
13	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	8.23	(1.13)
14	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	8.71	(2.94)
15	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	9.1	(1.64)
16	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	8.57	(0.98)
17	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	34.69	(25.88)
18	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	10.87	(5.47)
19	10	320.0	(0.0)	331.4	(100.18)	332.2	(99.15)	7.6	(1.34)
20	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	8.21	(1.36)
21	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	7.31	(1.02)
22	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	10.53	(7.19)
23	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	24.44	(18.41)
24	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	7.54	(0.97)
25	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	25.96	(17.15)
26	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	26.27	(21.25)
27	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	27.22	(27.95)
28	10	320.0	(0.0)	254.0	(131.68)	287.4	(103.8)	17.9	(14.96)
29	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	8.01	(0.91)
30	3	316.4	(2.76)	327.3	(56.62)	338.1	(49.29)	26.89	(23.29)
31	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	7.53	(1.64)
32	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	22.8	(14.95)
33	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	8.72	(1.61)
34	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	11.74	(7.71)
35	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	7.59	(1.28)
36	10	320.0	(0.0)	405.0	(0.0)	407.0	(0.0)	22.25	(22.74)
37	9	319.9	(0.32)	409.0	(0.0)	409.0	(0.0)	38.18	(22.01)
38	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	22.31	(16.33)
39	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	15.29	(17.51)
40	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	8.11	(1.6)
Summary	386	319.85	(0.9)	397.84	(45.66)	399.79	(38.74)	16.14	(16.65)

Table A.5: Results of $\mathcal{MM}$AS for the instances of size 509.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	400.0	(0.0)	505.0	(0.0)	507.0	(0.0)	45.83	(28.35)
2	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	54.97	(48.32)
3	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	13.32	(2.0)
4	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	37.61	(35.33)
5	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.15	(1.81)
6	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	24.87	(14.59)
7	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	47.89	(33.96)
8	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	18.82	(9.43)
9	10	400.0	(0.0)	505.0	(0.0)	507.0	(0.0)	17.97	(11.11)
10	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	13.77	(3.44)
11	9	399.9	(0.32)	481.5	(86.96)	482.3	(84.43)	44.18	(34.74)
12	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.27	(2.18)
13	5	397.0	(3.16)	309.1	(216.79)	396.9	(119.26)	88.47	(68.78)
14	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	39.3	(33.81)
15	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	36.08	(22.21)
16	3	397.1	(2.51)	377.4	(114.57)	451.4	(64.73)	126.49	(52.05)
17	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	13.09	(3.08)
18	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	34.8	(19.99)
19	10	400.0	(0.0)	353.8	(81.8)	355.4	(80.95)	72.4	(56.72)
20	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.4	(2.56)
21	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	13.52	(2.11)
22	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.33	(1.55)
23	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	25.58	(14.87)
24	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	22.99	(13.65)
25	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.06	(1.85)
26	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	45.97	(43.83)
27	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	18.62	(8.31)
28	9	399.8	(0.63)	400.1	(128.33)	407.1	(122.08)	64.09	(60.79)
29	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	78.73	(37.81)
30	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	38.33	(31.03)
31	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	60.47	(40.24)
32	9	399.9	(0.32)	488.9	(63.56)	495.6	(42.37)	74.45	(74.34)
33	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.21	(2.03)
34	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	81.15	(32.16)
35	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	24.48	(16.88)
36	9	399.9	(0.32)	507.0	(0.0)	508.0	(0.0)	83.81	(62.44)
37	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	17.72	(8.36)
38	9	399.5	(1.58)	490.1	(59.77)	498.4	(33.52)	63.9	(51.68)
39	1	396.6	(2.01)	305.4	(198.48)	413.0	(89.28)	111.68	(43.66)
40	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	15.98	(3.83)
Summary	373	399.74	(1.1)	487.01	(77.71)	494.53	(51.01)	41.57	(43.37)

Table A.6: Results of ACS for the instances of size 109.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.03)
2	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
3	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
4	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.03	(0.01)
5	8	79.8	(0.42)	109.0	(0.0)	109.0	(0.0)	0.09	(0.05)
6	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.02	(0.02)
7	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.05	(0.04)
8	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.02)
9	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.19	(0.52)
10	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
11	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.04)
12	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.02)
13	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.09	(0.2)
14	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.19	(0.11)
15	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.08	(0.15)
16	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.1	(0.23)
17	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
18	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.04	(0.04)
19	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
20	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.06	(0.06)
21	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.02	(0.01)
22	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.03	(0.02)
23	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.02	(0.02)
24	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.04	(0.02)
25	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.13	(0.3)
26	9	79.9	(0.32)	107.0	(0.0)	108.0	(0.0)	0.03	(0.03)
27	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.02	(0.01)
28	9	79.9	(0.32)	107.0	(0.0)	108.0	(0.0)	0.09	(0.16)
29	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.04	(0.03)
30	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.03)
31	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.05	(0.04)
32	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.03)
33	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.03)
34	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.02)
35	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.03)
36	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.03	(0.02)
37	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.02	(0.01)
38	9	79.9	(0.32)	109.0	(0.0)	109.0	(0.0)	0.02	(0.02)
39	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.02	(0.02)
40	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.04	(0.04)
Summary	388	79.97	(0.17)	108.4	(1.11)	108.7	(0.56)	0.05	(0.12)

Table A.7: Results of ACS for the instances of size 209.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.41	(0.42)
2	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.54	(0.46)
3	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.39	(0.23)
4	7	159.7	(0.48)	209.0	(0.0)	209.0	(0.0)	0.45	(0.35)
5	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	1.89	(2.65)
6	9	159.9	(0.32)	205.0	(0.0)	207.0	(0.0)	1.4	(2.34)
7	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	1.56	(1.75)
8	7	159.7	(0.48)	209.0	(0.0)	209.0	(0.0)	1.58	(1.74)
9	5	159.4	(0.7)	209.0	(0.0)	209.0	(0.0)	3.01	(3.01)
10	8	159.8	(0.42)	205.0	(0.0)	207.0	(0.0)	0.63	(0.33)
11	8	159.6	(0.97)	209.0	(0.0)	209.0	(0.0)	0.31	(0.24)
12	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.49	(0.56)
13	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.41	(0.25)
14	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	0.85	(0.93)
15	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	0.43	(0.33)
16	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.43	(0.63)
17	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	2.0	(3.1)
18	5	158.2	(3.99)	201.2	(24.67)	206.2	(8.85)	1.41	(1.73)
19	3	157.8	(1.62)	156.0	(43.89)	186.4	(18.59)	1.48	(2.71)
20	8	159.0	(2.16)	191.2	(40.27)	202.8	(13.31)	1.81	(2.75)
21	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	0.45	(0.55)
22	10	160.0	(0.0)	200.7	(4.35)	205.9	(1.45)	1.26	(2.76)
23	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	2.48	(2.71)
24	8	159.5	(1.27)	193.2	(49.96)	199.2	(30.99)	1.72	(3.24)
25	8	159.8	(0.42)	205.0	(0.0)	207.0	(0.0)	0.5	(0.49)
26	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	1.21	(2.29)
27	4	157.5	(2.17)	199.2	(8.46)	201.8	(6.2)	0.49	(0.3)
28	8	159.8	(0.42)	209.0	(0.0)	209.0	(0.0)	0.61	(0.54)
29	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.24	(0.15)
30	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.75	(0.81)
31	8	159.8	(0.42)	209.0	(0.0)	209.0	(0.0)	2.28	(3.76)
32	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.37	(0.38)
33	9	159.9	(0.32)	207.0	(0.0)	208.0	(0.0)	0.39	(0.53)
34	3	158.1	(1.45)	80.7	(95.95)	140.6	(47.43)	3.02	(2.82)
35	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	1.44	(1.95)
36	9	159.9	(0.32)	206.6	(1.26)	207.6	(1.26)	1.05	(1.95)
37	0	156.4	(1.58)	71.0	(58.96)	161.6	(16.98)	2.32	(1.27)
38	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.73	(0.98)
39	6	158.7	(1.95)	184.5	(51.68)	195.1	(30.44)	2.73	(3.3)
40	9	159.9	(0.32)	209.0	(0.0)	209.0	(0.0)	0.38	(0.19)
Summary	324	159.54	(1.27)	198.41	(37.19)	204.21	(16.97)	1.15	(1.92)

Table A.8: Results of ACS for the instances of size 309.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	7	239.7	(0.48)	309.0	(0.0)	309.0	(0.0)	16.57	(16.45)
2	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	20.57	(14.89)
3	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	15.47	(18.6)
4	7	239.7	(0.48)	307.0	(0.0)	308.0	(0.0)	13.91	(17.02)
5	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	11.94	(17.1)
6	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	11.68	(15.12)
7	7	238.4	(3.34)	300.9	(17.12)	301.3	(16.23)	7.69	(9.06)
8	9	239.8	(0.63)	307.0	(0.0)	308.0	(0.0)	3.92	(4.7)
9	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	1.73	(1.83)
10	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	10.46	(16.08)
11	7	238.8	(2.57)	279.0	(85.8)	289.2	(54.4)	8.41	(9.37)
12	9	239.9	(0.32)	309.0	(0.0)	309.0	(0.0)	6.06	(8.47)
13	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	9.32	(11.83)
14	9	239.9	(0.32)	309.0	(0.0)	309.0	(0.0)	6.12	(9.3)
15	3	237.9	(2.02)	298.8	(16.5)	305.6	(5.5)	19.54	(15.97)
16	9	239.9	(0.32)	309.0	(0.0)	309.0	(0.0)	6.85	(8.97)
17	6	237.8	(4.16)	281.4	(71.73)	294.5	(35.55)	9.8	(13.11)
18	8	239.7	(0.67)	309.0	(0.0)	309.0	(0.0)	9.93	(15.39)
19	10	240.0	(0.0)	153.8	(81.8)	164.2	(76.32)	11.31	(14.97)
20	8	239.6	(0.84)	309.0	(0.0)	309.0	(0.0)	15.16	(18.71)
21	9	239.8	(0.63)	307.0	(0.0)	308.0	(0.0)	6.59	(10.91)
22	9	239.9	(0.32)	309.0	(0.0)	309.0	(0.0)	14.04	(19.42)
23	8	239.8	(0.42)	303.0	(0.0)	306.0	(0.0)	15.8	(15.77)
24	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	11.08	(13.38)
25	7	239.1	(1.91)	290.7	(57.87)	291.6	(55.02)	20.34	(17.79)
26	9	239.9	(0.32)	307.0	(0.0)	308.0	(0.0)	6.9	(9.9)
27	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	8.74	(10.59)
28	3	237.7	(2.36)	272.9	(67.47)	281.9	(45.91)	13.15	(13.9)
29	7	239.7	(0.48)	309.0	(0.0)	309.0	(0.0)	15.55	(16.14)
30	6	239.4	(0.97)	309.0	(0.0)	309.0	(0.0)	7.59	(11.46)
31	8	239.8	(0.42)	309.0	(0.0)	309.0	(0.0)	5.41	(8.0)
32	2	237.2	(1.99)	245.5	(78.36)	277.8	(37.66)	14.36	(11.76)
33	9	239.9	(0.32)	280.2	(91.07)	298.7	(32.57)	8.28	(13.31)
34	9	239.8	(0.63)	308.8	(0.63)	308.8	(0.63)	9.46	(11.78)
35	7	239.7	(0.48)	265.7	(63.28)	294.1	(20.77)	6.52	(9.32)
36	4	239.4	(0.52)	305.0	(2.11)	306.0	(2.11)	8.51	(11.71)
37	9	239.9	(0.32)	305.0	(0.0)	307.0	(0.0)	13.47	(15.5)
38	9	239.9	(0.32)	309.0	(0.0)	309.0	(0.0)	12.93	(17.0)
39	4	238.2	(2.1)	290.2	(43.81)	295.6	(28.58)	17.09	(18.47)
40	8	239.8	(0.42)	307.0	(0.0)	308.0	(0.0)	2.09	(3.7)
Summary	306	239.47	(1.4)	297.3	(42.62)	301.01	(31.23)	10.86	(13.72)

Table A.9: Results of ACS for the instances of size 409.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	3	318.3	(2.5)	392.5	(39.68)	394.8	(36.15)	56.09	(32.05)
2	7	319.5	(0.85)	407.0	(0.0)	408.0	(0.0)	32.99	(33.8)
3	7	319.6	(0.7)	409.0	(0.0)	409.0	(0.0)	16.41	(24.21)
4	5	319.0	(1.33)	399.3	(30.67)	399.6	(29.73)	19.35	(23.08)
5	7	319.6	(0.7)	409.0	(0.0)	409.0	(0.0)	38.08	(32.49)
6	9	319.9	(0.32)	409.0	(0.0)	409.0	(0.0)	36.73	(32.01)
7	6	319.3	(1.06)	409.0	(0.0)	409.0	(0.0)	19.49	(28.92)
8	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	15.77	(22.66)
9	0	313.2	(0.63)	238.4	(0.52)	286.0	(0.0)	38.59	(24.1)
10	2	317.8	(2.35)	307.7	(143.79)	366.1	(55.65)	52.66	(28.57)
11	5	317.3	(4.3)	277.1	(174.6)	316.9	(121.29)	51.79	(38.3)
12	6	319.6	(0.52)	409.0	(0.0)	409.0	(0.0)	34.02	(28.51)
13	7	319.7	(0.48)	407.0	(0.0)	408.0	(0.0)	22.35	(31.1)
14	8	319.6	(0.97)	387.7	(61.03)	389.1	(59.77)	36.37	(37.44)
15	1	317.9	(1.2)	400.0	(4.83)	405.9	(1.45)	19.34	(16.68)
16	7	319.7	(0.48)	407.0	(0.0)	408.0	(0.0)	33.89	(36.26)
17	5	318.5	(1.78)	377.4	(39.95)	387.8	(26.54)	42.46	(44.74)
18	6	319.5	(0.71)	407.0	(0.0)	408.0	(0.0)	45.24	(39.25)
19	6	319.4	(0.84)	301.5	(114.93)	305.7	(109.91)	29.81	(29.63)
20	7	319.4	(1.26)	409.0	(0.0)	409.0	(0.0)	54.5	(30.58)
21	7	319.6	(0.7)	409.0	(0.0)	409.0	(0.0)	13.06	(12.97)
22	9	319.8	(0.63)	393.9	(47.75)	393.9	(47.75)	31.06	(32.71)
23	7	319.1	(1.66)	356.4	(112.18)	367.9	(87.24)	38.7	(27.22)
24	9	319.9	(0.32)	407.0	(0.0)	408.0	(0.0)	19.08	(29.1)
25	1	316.4	(2.01)	231.7	(124.92)	288.3	(66.77)	43.2	(31.2)
26	3	314.4	(3.89)	361.3	(32.92)	370.2	(26.99)	37.08	(34.14)
27	3	316.5	(2.8)	253.0	(193.77)	347.4	(74.5)	36.03	(35.23)
28	5	318.7	(2.16)	158.0	(92.71)	225.9	(64.43)	56.63	(27.14)
29	8	319.8	(0.42)	409.0	(0.0)	409.0	(0.0)	26.71	(28.8)
30	0	313.4	(0.7)	297.6	(5.06)	315.1	(9.8)	46.43	(34.52)
31	8	319.8	(0.42)	409.0	(0.0)	409.0	(0.0)	36.69	(35.91)
32	6	318.4	(2.91)	357.5	(110.72)	378.8	(63.61)	36.2	(35.07)
33	5	319.3	(0.82)	407.0	(0.0)	408.0	(0.0)	43.83	(39.14)
34	8	319.7	(0.67)	378.4	(64.51)	379.0	(63.25)	40.2	(33.28)
35	6	319.6	(0.52)	407.0	(0.0)	408.0	(0.0)	14.54	(12.61)
36	8	319.5	(1.08)	352.8	(110.05)	371.6	(74.63)	6.51	(9.09)
37	3	317.5	(3.27)	358.6	(86.59)	367.0	(70.41)	43.27	(32.04)
38	4	319.2	(0.92)	409.0	(0.0)	409.0	(0.0)	45.2	(33.39)
39	5	319.5	(0.53)	409.0	(0.0)	409.0	(0.0)	37.66	(38.42)
40	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	23.72	(34.12)
Summary	229	318.67	(2.28)	368.62	(89.3)	380.73	(60.54)	34.29	(32.19)

Table A.10: Results of ACS for the instances of size 509.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	6	399.4	(0.97)	503.0	(6.32)	505.6	(4.43)	97.53	(55.83)
2	0	393.6	(0.7)	111.9	(147.63)	337.0	(71.31)	68.73	(66.42)
3	8	399.7	(0.67)	509.0	(0.0)	509.0	(0.0)	75.35	(74.67)
4	0	397.3	(2.45)	364.4	(109.29)	368.2	(104.83)	104.17	(57.06)
5	5	399.5	(0.53)	509.0	(0.0)	509.0	(0.0)	78.45	(58.1)
6	6	398.3	(4.03)	491.3	(55.97)	492.1	(53.44)	68.84	(75.28)
7	5	398.6	(2.22)	484.1	(61.38)	487.9	(54.69)	125.92	(66.82)
8	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	55.28	(73.15)
9	8	399.6	(0.97)	471.6	(70.41)	484.2	(48.07)	75.06	(66.37)
10	8	399.8	(0.42)	509.0	(0.0)	509.0	(0.0)	81.37	(76.03)
11	2	397.8	(3.19)	307.6	(131.84)	318.5	(123.36)	91.27	(39.89)
12	8	399.6	(0.97)	509.0	(0.0)	509.0	(0.0)	68.53	(51.26)
13	0	394.6	(2.17)	183.4	(163.5)	311.6	(81.29)	99.58	(60.93)
14	3	397.5	(2.12)	303.1	(181.4)	330.3	(155.4)	62.99	(44.46)
15	4	397.3	(3.65)	459.4	(144.27)	485.2	(66.9)	101.59	(68.9)
16	0	393.9	(4.28)	286.1	(129.85)	398.1	(81.38)	94.84	(58.89)
17	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	115.08	(76.99)
18	5	399.1	(1.2)	444.6	(131.55)	446.6	(129.44)	55.27	(55.45)
19	4	397.1	(5.78)	391.8	(124.56)	398.7	(119.3)	115.94	(59.45)
20	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	60.39	(45.79)
21	8	399.8	(0.42)	507.0	(0.0)	508.0	(0.0)	76.7	(71.6)
22	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	55.21	(62.92)
23	8	399.8	(0.42)	509.0	(0.0)	509.0	(0.0)	71.15	(84.07)
24	7	399.7	(0.48)	509.0	(0.0)	509.0	(0.0)	42.91	(44.73)
25	7	399.6	(0.7)	509.0	(0.0)	509.0	(0.0)	48.58	(57.28)
26	7	399.6	(0.7)	509.0	(0.0)	509.0	(0.0)	59.6	(60.43)
27	7	399.5	(0.97)	507.2	(5.69)	507.8	(3.79)	81.95	(66.91)
28	1	397.9	(1.1)	316.3	(160.69)	355.6	(127.37)	80.97	(64.44)
29	3	396.1	(5.53)	481.9	(51.36)	485.6	(45.03)	93.8	(64.79)
30	3	399.2	(0.63)	437.6	(150.52)	478.6	(64.09)	103.58	(56.87)
31	8	399.5	(1.27)	474.5	(109.1)	494.9	(44.59)	65.28	(55.88)
32	3	398.5	(1.51)	435.0	(119.37)	454.3	(74.11)	134.59	(68.93)
33	9	399.9	(0.32)	509.0	(0.0)	509.0	(0.0)	84.75	(75.99)
34	3	398.7	(1.34)	507.2	(2.9)	507.2	(2.9)	87.37	(48.32)
35	5	398.2	(2.74)	431.3	(128.56)	436.6	(119.35)	106.18	(55.39)
36	7	399.6	(0.7)	506.6	(1.26)	507.6	(1.26)	95.17	(69.16)
37	8	399.8	(0.42)	509.0	(0.0)	509.0	(0.0)	52.86	(62.93)
38	8	398.8	(3.16)	486.2	(72.1)	496.0	(41.11)	91.92	(70.39)
39	2	396.1	(2.47)	238.4	(193.77)	387.5	(86.92)	84.24	(59.32)
40	9	399.8	(0.63)	509.0	(0.0)	509.0	(0.0)	58.13	(63.83)
Summary	221	398.56	(2.63)	444.16	(132.26)	465.47	(87.94)	81.78	(63.8)

algorithm, and Smith-Waterman algorithm respectively). Finally, the sixth column—*time*—provides the average computation time for solving an instance (in seconds).

Table A.11: Results of ML-ACO for the instances of size 109.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
2	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
3	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
4	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
5	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
6	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
7	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
8	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
9	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
10	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
11	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
12	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
13	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.01	(0.01)
14	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
15	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.01	(0.01)
16	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
17	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
18	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.01	(0.01)
19	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
20	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
21	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.0)
22	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
23	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
24	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
25	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
26	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
27	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
28	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
29	10	80.0	(0.0)	105.0	(0.0)	107.0	(0.0)	0.0	(0.01)
30	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
31	10	80.0	(0.0)	107.0	(0.0)	108.0	(0.0)	0.0	(0.01)
32	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
33	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.0)
34	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
35	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.0)
36	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.0)
37	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
38	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.01)
39	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.01	(0.0)
40	10	80.0	(0.0)	109.0	(0.0)	109.0	(0.0)	0.0	(0.01)
Summary	400	80.0	(0.0)	108.4	(1.11)	108.7	(0.56)	0.0	(0.01)

Table A.12: Results of ML-ACO for the instances of size 209.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.0)
2	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
3	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
4	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
5	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
6	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	0.02	(0.0)
7	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
8	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	3.6	(0.5)
9	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.1	(0.0)
10	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	0.07	(0.0)
11	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
12	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
13	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.01	(0.01)
14	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
15	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.01	(0.01)
16	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
17	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
18	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.03	(0.01)
19	2	158.5	(0.85)	160.8	(33.17)	180.5	(19.57)	3.63	(3.28)
20	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.01)
21	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
22	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.01)
23	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
24	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
25	10	160.0	(0.0)	205.0	(0.0)	207.0	(0.0)	0.02	(0.0)
26	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
27	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
28	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
29	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
30	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.01)
31	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
32	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.01)
33	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.0)
34	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
35	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.03	(0.0)
36	10	160.0	(0.0)	207.0	(0.0)	208.0	(0.0)	0.02	(0.01)
37	1	157.9	(0.88)	188.0	(12.94)	195.7	(7.62)	6.08	(2.53)
38	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
39	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.03	(0.02)
40	10	160.0	(0.0)	209.0	(0.0)	209.0	(0.0)	0.02	(0.0)
Summary	383	159.91	(0.44)	206.67	(9.71)	207.66	(5.78)	0.36	(1.36)

Table A.13: Results of ML-ACO for the instances of size 309.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.04	(0.01)
2	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.04	(0.01)
3	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.04	(0.0)
4	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.04	(0.0)
5	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
6	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
7	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.07	(0.02)
8	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.04	(0.0)
9	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.04	(0.01)
10	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.04	(0.01)
11	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.1	(0.06)
12	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	9.42	(1.6)
13	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
14	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.07	(0.03)
15	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
16	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	1.03	(0.21)
17	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
18	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.04	(0.01)
19	10	240.0	(0.0)	115.0	(0.0)	128.0	(0.0)	0.04	(0.0)
20	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.06	(0.01)
21	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.06	(0.01)
22	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.0)
23	10	240.0	(0.0)	303.0	(0.0)	306.0	(0.0)	0.05	(0.01)
24	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.06	(0.01)
25	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
26	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.05	(0.01)
27	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.04	(0.01)
28	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
29	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
30	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.0)
31	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.06	(0.02)
32	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
33	10	240.0	(0.0)	21.0	(0.0)	206.0	(0.0)	0.07	(0.02)
34	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
35	10	240.0	(0.0)	226.4	(67.65)	281.2	(22.21)	0.05	(0.01)
36	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.05	(0.01)
37	10	240.0	(0.0)	305.0	(0.0)	307.0	(0.0)	0.05	(0.01)
38	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.07	(0.01)
39	10	240.0	(0.0)	309.0	(0.0)	309.0	(0.0)	0.05	(0.01)
40	10	240.0	(0.0)	307.0	(0.0)	308.0	(0.0)	0.05	(0.0)
Summary	400	240.0	(0.0)	294.24	(55.5)	300.88	(32.43)	0.31	(1.49)

Table A.14: Results of ML-ACO for the instances of size 409.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.08	(0.01)
2	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.07	(0.01)
3	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.08	(0.01)
4	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	1.95	(0.15)
5	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.08	(0.01)
6	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.08	(0.01)
7	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.1	(0.02)
8	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.11	(0.04)
9	2	316.4	(2.22)	322.0	(87.51)	347.9	(62.98)	44.81	(28.06)
10	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
11	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.17	(0.07)
12	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
13	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.01)
14	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.01)
15	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.01)
16	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.08	(0.0)
17	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
18	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.81	(0.08)
19	10	320.0	(0.0)	215.0	(0.0)	217.0	(0.0)	0.09	(0.0)
20	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
21	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
22	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.1	(0.01)
23	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.11	(0.01)
24	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.01)
25	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	1.08	(0.29)
26	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.0)
27	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
28	10	320.0	(0.0)	228.5	(123.18)	267.3	(97.09)	22.79	(8.55)
29	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.0)
30	2	315.2	(2.53)	318.6	(47.64)	331.4	(40.9)	24.49	(14.88)
31	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.1	(0.01)
32	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.0)
33	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.09	(0.0)
34	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.11	(0.01)
35	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.1	(0.01)
36	10	320.0	(0.0)	405.0	(0.0)	407.0	(0.0)	0.09	(0.01)
37	9	319.7	(0.95)	409.0	(0.0)	409.0	(0.0)	40.99	(28.27)
38	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
39	10	320.0	(0.0)	409.0	(0.0)	409.0	(0.0)	0.09	(0.01)
40	10	320.0	(0.0)	407.0	(0.0)	408.0	(0.0)	0.1	(0.01)
Summary	383	319.78	(1.06)	394.55	(50.13)	396.87	(43.18)	3.5	(12.28)

Table A.15: Results of ML-ACO for the instances of size 509.

Instance	Solved	Quality		Global		Local		Time (sec)	
1	10	400.0	(0.0)	505.0	(0.0)	507.0	(0.0)	5.29	(1.2)
2	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.03)
3	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.15	(0.01)
4	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	5.75	(2.85)
5	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.02)
6	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	8.15	(1.69)
7	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.15	(0.01)
8	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.17	(0.01)
9	10	400.0	(0.0)	505.0	(0.0)	507.0	(0.0)	0.23	(0.05)
10	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.17	(0.03)
11	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
12	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
13	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.61	(0.03)
14	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.21	(0.06)
15	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	36.45	(4.55)
16	0	397.7	(0.95)	392.6	(79.14)	469.7	(26.44)	78.54	(68.81)
17	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.18	(0.02)
18	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	0.17	(0.03)
19	9	399.9	(0.32)	403.1	(96.58)	407.1	(96.59)	106.21	(51.33)
20	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	3.43	(0.24)
21	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	0.15	(0.01)
22	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
23	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
24	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.17	(0.02)
25	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.21	(0.06)
26	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.21	(0.07)
27	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.17	(0.01)
28	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	0.17	(0.01)
29	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.25	(0.14)
30	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
31	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
32	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.18	(0.02)
33	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.17	(0.01)
34	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
35	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	5.86	(0.68)
36	10	400.0	(0.0)	507.0	(0.0)	508.0	(0.0)	0.16	(0.01)
37	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.16	(0.01)
38	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	0.22	(0.05)
39	1	396.8	(1.81)	348.1	(170.84)	442.0	(59.24)	26.98	(56.4)
40	10	400.0	(0.0)	509.0	(0.0)	509.0	(0.0)	33.52	(2.84)
Summary	380	399.86	(0.68)	499.02	(46.7)	503.6	(26.23)	7.9	(26.42)

Bibliography

[1] W. Bains and G. C. Smith. A novel method for nucleid acid sequence determination. *Journal of Theoretical Biology*, 135:303–307, 1988.

[2] J. Błażewicz, P. Formanowicz, F. Guinand, and M. Kasprzak. A heuristic managing errors for DNA sequencing. *Bioinformatics*, 18(5):652–660, 2002.

[3] J. Błażewicz, P. Formanowicz, M. Kasprzak, W. T. Markiewicz, and J. Weglarz. DNA sequencing with positive and negative errors. *Journal of Computational Biology*, 6:113–123, 1999.

[4] J. Błażewicz, P. Formanowicz, M. Kasprzak, W. T. Markiewicz, and J. Weglarz. Tabu search for DNA sequencing with false negatives and false positives. *European Journal of Operational Research*, 125:257–265, 2000.

[5] J. Błażewicz, F. Glover, and M. Kasprzak. DNA sequencing—Tabu and scatter search combined. *INFORMS Journal on Computing*, 16(3):232–240, 2004.

[6] J. Błażewicz, F. Glover, and M. Kasprzak. Evolutionary approaches to DNA sequencing with errors. *Annals of Operations Research*, 138:67–78, 2005.

[7] J. Błażewicz and M. Kasprzak. Complexity of DNA sequencing by hybridization. *Theoretical Computer Science*, 290(3):1459–1473, 2003.

[8] J. Błażewicz, M. Kasprzak, and W. Kuroczycki. Hybrid genetic algorithm for DNA sequencing with errors. *Journal of Heuristics*, 8:495–502, 2002.

[9] C. Blum and M. Dorigo. The hyper-cube framework for ant colony optimization. *tsmcb*, 34(2):1161–1172, 2004.

[10] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[11] C. Blum and M. Yábar Vallès. New constructive heuristics for DNA sequencing by hybridization. In *Proceedings of WABI 2006 – 6th International Workshop on Algorithms in Bioinformatics*, Lecture Notes in Bioinformatics (LNBI/LNCS). Springer-Verlag, Berlin, Germany, 2006. To appear.

[12] E. Bonabeau, G. Theraulaz, J.-L. Deneubourg, S. Aron, and S. Camazine. Self-organization in social insects. In *Trends in Ecol. Evol. 12*, pages 188–193. 1997.

[13] A. Brandt. Multilevel computations: Review and recent developments. In S. F. McCormick, editor, *Multigrid Methods: Theory, Applications, and Supercomputing, Proceedings of the 3rd Copper Mountain Conference on Multigrid Methods*, volume 110 of *Lecture Notes in Pure and Applied Mathematics*, pages 35–62. Marcel Dekker, New York, 1988.

[14] T. N. Bui and W. A. Youssef. An enhanced genetic algorithm for DNA sequencing by hybridization with positive and negative errors. In K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. K. Burke, P. J. Darwen, D. Dasgupta, D. Floreano, J. A. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens, and A. M. Tyrrell, editors, *Proceedings of the GECCO 2004 – Genetic and Evolutionary Computation Conference*, volume 3103 of *Lecture Notes in Computer Science*, pages 908–919. Springer Verlag, Berlin, Germany, 2004.

[15] V. Cerny. A thermodynamical approach to the traveling saleman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[16] J.L Deneubourg, S. Aron, S. Goss, and J.M. Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior 3*, pages 159–168, 1990.

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

[18] M. Dorigo, G. Di Caro, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, pages 137–172, 1999.

[19] M. Dorigo and L. M. Gambardella. Ant colony system: A cooperative learning approach to the travelling salesman problem. *IEEE Transactions on Evolutionary Computatiom*, 1:53–56, 1997.

[20] M. Dorigo, V. Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[21] M. Dorigo and T. Stützle. *Ant Colony Optimization (Bradford Books)*. The MIT Press, 2004.

[22] R. Drmanac, I. Labat, R. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: Theory of the method. *Genomics*, 4:114–128, 1989.

[23] T. A. Endo. Probabilistic nucleotide assembling method for sequencing by hybridization. *Bioinformatics*, 20(14):2181–2188, 2004.

[24] E. R. Fernandes and C. C. Ribeiro. Using an adaptive memory strategy to improve a multistart heuristic for sequencing by hybridization. In S. E. Nikoletseas, editor, *Proceedings of WEA 2005 – 4th International Workshop on Experimental and Efficient Algorithms*, volume 3503 of *Lecture Notes in Computer Science*, pages 4–15. Springer Verlag, Berlin, Germany, 2005.

[25] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[26] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[28] Y. P. Lysov, V. L. Florentiev, A. A. Khorlin, K. R. Khrapko, and V. V. Shik. Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. a new method. *Doklady Akademii nauk SSSR*, 303:1508–1511, 1988.

[29] S. B Needleman and C. D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, pages 443–453, 1970.

[30] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall Inc., 1982.

[31] P. A. Pevzner. l-tuple DNA sequencing: Computer analysis. *Journal of Biomulecular Structure and Dynamics*, 7:63–73, 1989.

[32] T. F Smith and M. S Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, pages 195–197, 1981.

[33] T. Stützle and H. Hoos. MAX–MIN ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[34] C. Walshaw. A multilevel approach to the travelling salesman problem. *Operations Research*, 50(5):862–877, 2002.

[35] C. Walshaw. Multilevel refinement for combinatorial optimization problems. *Annals of Operations Research*, 131:325–372, 2004.

[36] C. Walshaw and M. Cross. Mesh partitioning: A multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.