Advanced search and data management services in Life Sciences

PhD Thesis

of

Thanasis Vergoulis

Diploma in Computer Engineering and Informatics, & Univ. of Patras (2007)

Athens, August 2014
Προηγμένες υπηρεσίες αναζήτησης και διαχείρισης δεδομένων στις Βιοεπιστήμες

Διδακτορική Διατριβή
tου
Αθανασίου Βεργούλη
Διπλωματούχου Μηχανικού Υπολογιστών & Πληροφορικής Παν. Πατρών (2007)

Συμβουλευτική Επιτροπή: Τ. Σελλής
Ι. Βασιλείου
Θ. Δαλαμάγκας

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 7η Αυγούστου 2014.

Τ. Σελλής Καθ. ΕΜΠ
Ι. Βασιλείου Καθ. ΕΜΠ
Θ. Δαλαμάγκας Ερευνητής Β' Ερ. Κέντρου ΑΘΗΝΑ

Δ. Γουνόπουλος Καθ. ΕΚΠΑ
Δ. Φωτάκης Αναπ. Καθ. ΕΜΠ
Ν. Κοζύρης Καθ. ΕΜΠ

Α. Δεληγιαννάκης Επ. Καθ. Πολ. Κρήτης

Αθήνα, Αύγουστος 2014
Thanasis Vergoulis
Computer Engineer and Informatics engineer, PhD, N.T.U.A.

© 2014 - All rights reserved

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ο- αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή εφευρετικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Η έγκριση της διδακτορικής διατριβής από την Ανώτατη Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Τηλεπικοινωνιών του Ε. Μ. Πολυτεχνείου δεν υποδηλώνει αποδοχή των γνωμών του συγγραφέα (N. 5343/1932, Άρθρο 202).
Contents

1 Introduction ............................................. 1
   1.1 Contributions ......................................... 2
   1.2 Outline .............................................. 4

2 Preliminaries ........................................... 5
   2.1 Management of sequence databases ...................... 5
      2.1.1 Sequence matching problems ....................... 6
         2.1.1.1 Exact sequence matching ...................... 6
         2.1.1.2 Approximate sequence matching ................ 7
         2.1.1.3 Algorithms and index structures for ASM .... 7
   2.2 Data management and processing in Life Sciences .... 10
      2.2.1 Concepts from biology ............................ 10
         2.2.1.1 DNA, genome, and genes ...................... 10
         2.2.1.2 Protein synthesis and messenger RNA .......... 11
         2.2.1.3 MicroRNAs .................................... 12
      2.2.2 Computational methods for microRNA target prediction . . 13
         2.2.2.1 DIANA microT v.3 ................................ 13
         2.2.2.2 DIANA microT v.4 ................................ 14
         2.2.2.3 DIANA microT v.5 ................................ 14
      2.2.3 DNA read alignment ................................ 14

3 Efficient miRNA target prediction ......................... 17
   3.1 Efficient sequence matching for target prediction .... 17
      3.1.1 Motivation and contribution ...................... 17
      3.1.2 The ARSM problem ................................ 19
      3.1.3 ARSM Characteristics .............................. 21
         3.1.3.1 Overlapping Occurrences ...................... 21
         3.1.3.2 Prefix and Suffix Expansions ................. 23
      3.1.4 The PS-ARSM Method ............................... 25
         3.1.4.1 Region Lattice ................................ 26
         3.1.4.2 Algorithm Description ......................... 26
         3.1.4.3 Implementation ............................... 29
      3.1.5 Cost Analysis and Optimization .................... 34
      3.1.6 Experimental evaluation .......................... 36
         3.1.6.1 Setup ........................................ 36
         3.1.6.2 Results ...................................... 37
   3.2 Cloud-based target prediction systems .................. 41
      3.2.1 TarCloud: target prediction on MS Azure ........ 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.1.1 Technologies and system architecture</td>
<td>41</td>
</tr>
<tr>
<td>3.2.1.2 User interface and evaluation</td>
<td>43</td>
</tr>
<tr>
<td>3.2.2 MR-microT: target prediction using MapReduce</td>
<td>44</td>
</tr>
<tr>
<td>3.2.2.1 Motivation</td>
<td>44</td>
</tr>
<tr>
<td>3.2.2.2 Technologies and system architecture</td>
<td>45</td>
</tr>
<tr>
<td>3.2.2.3 User interface and evaluation</td>
<td>47</td>
</tr>
<tr>
<td>3.3 Conclusions</td>
<td>49</td>
</tr>
<tr>
<td>4 Research infrastructures for miRNA research</td>
<td>51</td>
</tr>
<tr>
<td>4.1 Needs for miRNA tools and data</td>
<td>51</td>
</tr>
<tr>
<td>4.2 DIANA microT Web server: Searching for miRNA predicted targets</td>
<td>52</td>
</tr>
<tr>
<td>4.2.1 DIANA microT v.3</td>
<td>53</td>
</tr>
<tr>
<td>4.2.2 DIANA microT v.4</td>
<td>54</td>
</tr>
<tr>
<td>4.2.3 DIANA microT v.5</td>
<td>57</td>
</tr>
<tr>
<td>4.3 DIANA miRGen: Revealing information about miRNA transcripts</td>
<td>59</td>
</tr>
<tr>
<td>4.3.1 DIANA miRGen v.2</td>
<td>59</td>
</tr>
<tr>
<td>4.4 DIANA TarBase: Searching for experimentally verified miRNA targets</td>
<td>63</td>
</tr>
<tr>
<td>4.4.1 Related work</td>
<td>64</td>
</tr>
<tr>
<td>4.4.2 DIANA TarBase v.6</td>
<td>65</td>
</tr>
<tr>
<td>4.4.3 Automatic extraction of miRNA targets: towards DIANA TarBase v.7</td>
<td>69</td>
</tr>
<tr>
<td>4.5 DIANA mirPath: Discovering the role of miRNAs in pathways</td>
<td>72</td>
</tr>
<tr>
<td>4.6 DIANA mirPub: Searching for miRNA-related publications</td>
<td>75</td>
</tr>
<tr>
<td>4.6.1 DIANA mirPub description</td>
<td>76</td>
</tr>
<tr>
<td>4.6.1.1 miRNA-to-publication associations</td>
<td>76</td>
</tr>
<tr>
<td>4.6.1.2 Capturing miRNA data evolution</td>
<td>77</td>
</tr>
<tr>
<td>4.6.1.3 Features and interface</td>
<td>79</td>
</tr>
<tr>
<td>4.6.1.3.1 miRNA publication search</td>
<td>79</td>
</tr>
<tr>
<td>4.6.1.3.2 Visualizing miRNA data evolution</td>
<td>80</td>
</tr>
<tr>
<td>4.6.1.3.3 User contributed data</td>
<td>81</td>
</tr>
<tr>
<td>4.6.2 Evaluation</td>
<td>81</td>
</tr>
<tr>
<td>4.6.2.1 miRNA publications and data evolution statistics</td>
<td>81</td>
</tr>
<tr>
<td>4.6.2.2 Evaluating literature retrieval</td>
<td>83</td>
</tr>
<tr>
<td>4.7 Conclusions</td>
<td>84</td>
</tr>
<tr>
<td>5 Sequence alignment of long DNA reads</td>
<td>87</td>
</tr>
<tr>
<td>5.1 Background</td>
<td>87</td>
</tr>
<tr>
<td>5.1.1 Useful notations and definitions</td>
<td>87</td>
</tr>
<tr>
<td>5.1.2 The pigeonhole principle</td>
<td>88</td>
</tr>
<tr>
<td>5.1.3 Our motivation</td>
<td>89</td>
</tr>
<tr>
<td>5.2 The Hitmap index</td>
<td>90</td>
</tr>
<tr>
<td>5.2.1 Basic Concepts</td>
<td>90</td>
</tr>
<tr>
<td>5.2.2 Index Structure</td>
<td>93</td>
</tr>
<tr>
<td>5.2.3 Index compression</td>
<td>94</td>
</tr>
<tr>
<td>5.3 Querying the Hitmap Index</td>
<td>94</td>
</tr>
<tr>
<td>5.3.1 The Hitmap algorithm</td>
<td>94</td>
</tr>
<tr>
<td>5.3.1.1 Efficient bitwise operations to find candidates</td>
<td>96</td>
</tr>
<tr>
<td>5.3.2 Optimising bitwise operations</td>
<td>99</td>
</tr>
<tr>
<td>5.4 Evaluation</td>
<td>99</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 Functions of miRNAs. ................................................................. 13

3.1 An example of an ARSM problem. ............................................. 18
3.2 An approximate occurrence of GATTACA in AACCGAATTACACC. .... 20
3.3 ARSM example. ............................................................................ 21
3.4 The region lattice for the ARSM example. ................................. 26
3.5 Core occurrences, head occurrences and seeds for suffix chains $c_1$, $c_2$, and $c_3$. ................................................................. 27
3.6 Seeds, candidates and temporal results. ................................. 29
3.7 The PS-ARSM algorithm. .............................................................. 30
3.8 An excerpt of the head_occs hash table, containing the head occurrences of suffix chain $c_2$. ................................................................. 31
3.9 An excerpt of the cands hash table, containing the candidates for $R3$, $R5$, and $R7$. ................................................................. 32
3.10 Method pExp updates the add_cost array of head_occs entry $h$. ... 33
3.11 Example of vertical and horizontal lattice split. ....................... 35
3.12 Varying the pattern sequence length $|P|$. ................................. 38
3.13 Varying the data sequence length $|S|$. ...................................... 38
3.14 Varying the ratio of core to pattern length $|C|/|P|$. .................... 39
3.15 Varying the core position $cPos$. .............................................. 39
3.16 Varying the cost ratio $\alpha$. .................................................... 40
3.17 Varying the alphabet size $\Sigma$. ............................................. 40
3.18 Setting with real patterns. ....................................................... 41
3.19 The workflow of TarCloud jobs. ............................................. 42
3.20 The TarCloud architecture. .................................................... 42
3.21 A snapshot of the TARCLOUD search interface ..................... 44
3.22 A snapshot of the TARCLOUD results renderer for miRNA “my-miRNA-1”. ................................................................. 44
3.23 MR-microT system architecture ............................................ 46
3.24 Screenshot of MR-microT’s user interface. ............................. 48
3.25 Progress view of MR-microT, displaying the current status of the target prediction of a single miRNA; and Result view presenting the list of predicted targets ............................................ 49
4.1 A screenshot of DIANA microT v.3 interface while showing the predicted targets of miRNA “hsa-let-7a” in the gene “CCND1”. .... 54
4.2 A screenshot of DIANA microT v.4 interface while showing the predicted targets of miRNA “mmu-miR-455-star”. .................. 56
4.3 The data evolution history related to miRNA with name “mmu-miR-455*” .................................................. 57
4.4 Flowchart depicting an analysis pipeline directly available from the Web server interface........................................... 58
4.5 Part of the entity-relationship model of miRGen’s database .......... 61
4.6 Two screenshots of miRGen interface. One for a miRNA search (A) and the other for a transcription factor search (B) ......................... 62
4.7 The annual growth of miRNA-related publications in PubMed and the number of entries in miRBase database. ....................... 63
4.8 The adopted text-mining-assisted curation pipeline......................................................... 66
4.9 A screenshot of DIANA TarBase v.6 interface................................. 68
4.10 The process to identify miRNA-gene interactions in literature. ......... 70
4.11 A snapshot of DIANA mirPath interface........................................ 73
4.12 A snapshot of a KEGG pathway visualisation.............................. 74
4.13 An example heat map from the DIANA mirPath’s interface. ........... 75
4.14 The workflow of miRNA-to-publication associations extraction. ...... 76
4.15 The evolution timeline of term “hsa-mir-29b-1” (involving hairpins MI0000105 and MI0000107). ............................. 78
4.16 The evolution timeline of term “hsa-mir-98” (involving hairpin MI0000100). .............................................................. 78
4.17 Screenshot of mirPub’s interface............................................. 80
4.18 The number of NEW & DELETE changes introduced in each miRBase version. .......................................................... 83
4.19 The number of NAME, SEQUENCE, and NAME-SEQUENCE changes introduced in each miRBase version. ....................... 83

5.1 A sample data and query sequence along with the corresponding partitions. ................................................................. 88
5.2 Part of the Hitmap index of the data sequence presented in Figure 5.1. 93
5.3 The Hitmap algorithm......................................................... 96
5.4 An example set of bitsets, its t-occurrences bitset, and the counter bitsets used for the t-occurrences bitset computation (t = 3).............. 97
5.5 Pseudocode of an algorithm to calculate the t-occurrences bitset of a given set of bitsets .............................................. 98
5.6 Exec. time for Hitmap, WHAM, and RBSA for various alignment thresholds on SYNTH dataset, in case of large reads. .................. 101
5.7 Exec. time for Hitmap, WHAM, and RBSA for various alignment thresholds on SYNTH dataset, in case of small reads. .................. 102
5.8 Exec. time for Hitmap, and WHAM for various alignment thresholds on Homo sapiens’ chromosome 19, in case of large reads. ........ 103
List of Tables

3.1 Common notation.......................................................... 26
3.2 Experimental parameters.............................................. 36
3.3 Memory consumption of PS-ARSM hash tables....................... 37
3.4 Running time breakdown for PS-ARSM phases......................... 37
3.5 Measurements for TarCloud performance........................... 45
3.6 Execution time of MR-microT for various cluster sizes for human and mouse genome................................................... 49

4.1 Preliminary evaluation of the automated extraction of miRNA-gene interactions............................................................... 71
4.2 Comparison of several miRNA publication databases.............. 82
4.3 Some interesting statistics of mirPub’s publications............... 82
4.4 The major journals containing miRNA related publications and the major MeSH diseases related to miRNAs based on publications..... 82
4.5 Evaluation of mirPub’s effectiveness in miRNA literature retrieval in comparison to PubMed.................................................. 84
4.6 Evaluating how knowing miRNA data evolution improves searching for publications......................................................... 84

5.1 Hitmap configuration for the experiments on SYNTH dataset.. 102
5.2 Hitmap configuration for the experiments on chromosome 19 of Homo sapiens.............................................................. 103
PREFACE

This thesis is submitted in fulfilment of the requirements for the degree of Doctor of Philosophy, in the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Greece. The presented work describes methods for managing vast amounts of data from Life Sciences and has been carried out during the last six years in the Knowledge and Database Systems Laboratory (KDBSL) of NTUA and in the Institute for the Management of Information Systems (IMIS) of Research Center ‘Athena’.

I am grateful to Prof. Timos Sellis, Dr. Theodore Dalamagas, and Dr. Dimitris Sacharidis for their guidance and valuable advice. Their expertise in the field of data management was a valuable resource of ideas. Moreover, I would like to thank Prof. Artemis G. Hatzigeorgiou and her team for introducing me to the field of miRNA research and for responding kindly to each of my questions. I also appreciate the contribution of Nikos Kostoulas, Ilias Kanellos, and Rodothea-Myrsini Tsoupidi who worked on many issues related to this work.

Last but not least, I would like to provide my warmest thanks to all my colleagues in the KDBS Lab and IMIS for their help, cooperation, and the great time we spent together during the last six years.

Thanasis Vergoulis
Athens, July 2014
To my parents, my brother, and, of course, to Georgia.
ABSTRACT

The need for data management and processing approaches in *life sciences* is becoming more intense due to the continuous technological advances in the machines that produce data from biological samples. In today’s era, these machines produce vast amount of data that need to be processed. Most of these data are represented as *sequences* and their processing consists, mainly, of applying sequence alignment algorithms on them. State-of-the-art sequence alignment algorithms fail to perform efficiently for such *big data*, thus, the introduction of novel approaches is apparent. To make the condition worse, novel findings sometimes raise novel processing needs that cannot be fulfilled by adapting already existent approaches. Again, new methods are required. Finally, new rapidly evolving fields in life sciences, like that of miRNA research, lack centralised information resources. The knowledge in such fields is scattered in a multitude of scientific publications slowing down the work of researchers.
Chapter 1

Introduction

The life sciences comprise the fields of science that involve the study of living organisms and life mechanisms, in general. Studying mechanisms of life is performed mainly by conducting biochemical experiments that investigate possible interactions arising among the chemical compounds of the cells\textsuperscript{1}. However, such experiments are costly and time consuming. Therefore, investigating the possible interaction between two chemicals which are unrelated is an important waste of money and time. This is the reason why finding convenient computer representations of the cell compounds and introducing efficient computational analysis on them to reveal the most probable interactions became the subject of a new, broad field in life sciences, named Bioinformatics.

The need for data management and processing in life sciences is becoming more intense due to continuous technological development of the laboratory equipment that analyse biological samples to produce computer readable representations of the cell compounds. Since the most convenient is to represent these compounds as sequences, the aforementioned equipment is called sequencing machine. In today’s era, sequencing machines produce data with constantly increasing volumes. As a consequence, we have reached the point where the cost to produce the data is less than the one required to analyse them. Therefore, the need for efficient management of these big data from life sciences is becoming apparent.

Data management and processing approaches for sequences have been proposed from the early sixties. However, in the recent years, sequences from life sciences raised major issues that the state-of-the-art approaches were unable to resolve. The first issue is related to the above mentioned boost in the production of data to be analysed, guided by the recent advances in the technology of sequencing machines. A modern sequencing machine is capable to produce several GB of data per day and a typical run may endure for several days. The existing approaches have been proven to be inadequate, in terms of performance, for this vast amount of data waiting to be processed.

Another issue is created by the rise of a new generation of sequencing machines. Usually, biological sequences are long (for example, a typical DNA chromosome may contain more than 100M symbols), however, sequencing machines produce them in parts of particular size, called reads. Since early sequencing machines were producing small reads containing at most 100 symbols, the first generation of sequence processing approaches was optimised for sequences of this size. However, small read

\textsuperscript{1}Cells are the building blocks of all living organisms.
sizes can result in problems during the sequence processing. For example, the most common type of processing consists of aligning DNA reads inside a given genome sequence. The alignments of small reads can be dramatically affected by the presence of structural variations or single-nucleotide polymorphisms (SNPs) inside them. This motivated a new generation of sequencing machines that can produce much longer reads. However, to provide this benefit, this new generation of sequencing machines sacrifices accuracy, which means that their reads contain increased number of misplaced symbols. Existing sequence processing techniques have been proven to be inadequate in the case of long and less accurate reads. Therefore, the need for approaches that perform well under the aforementioned scenario becomes apparent.

Furthermore, novel findings about life mechanisms sometimes create novel data processing needs. In these cases, there are no state-of-the-art approaches and adapting existing methods may result in poor performance. Therefore, entirely new processing approaches must be developed. For example, in early 2000’s, the discovery of miRNAs and their role in deactivating genes motivated the development of computational methods that try to predict the gene-targets of each miRNA. The chemical binding between a miRNA and the transcript of a gene that results in the deactivation of the gene motivated the use of complex sequence alignment criteria by the most accurate of these prediction algorithms (like DIANA microT). However, adapting existing sequence alignment approaches to search for gene locations that satisfy the aforementioned criteria results in poor performance. Thus, more sophisticated approaches are required.

Finally, a major issue in life sciences is that although there is a plethora of repositories collecting and distributing interesting information of general interest (like those hosted by Ensembl and NCBI), there is an absence of similar resources for more specialised fields. For instance, this is the case in the very important field of miRNA research. Although a hub collecting some interesting information related to each identified miRNA molecule exists\(^2\), the relationship of miRNAs with genes, their role in metabolic pathways, their expression profiles, and many other information related to them are either scattered in relevant scientific publications or not existent at all (since additional analysis should be done to reveal them). However, miRNAs contribute in very crucial functions of life, being responsible for the deactivation of important genes. Gaining knowledge about miRNAs could help in the direction of understanding and treating important diseases, such as several types of cancer, Alzheimer’s disease, etc. This is why, Web repositories and tools that cover the aforementioned information gap would be very valuable.

### 1.1 Contributions

This dissertation presents various methods for managing data from life sciences. We focus on two important fields, miRNA target prediction and DNA read alignment. The former incorporates computational methods trying to reveal interactions between miRNAs and genes, while the latter involves aligning small DNA sequences in reference genome sequences, which is a useful preprocessing step for almost any important computational analysis for biological molecules. Furthermore, we investigate ways to assist research related to miRNA molecules by the dissemination of

\(^2\)http://www.mirbase.org
useful knowledge related to them. Our contributions involve the following.

1. We consider the problem of providing accurate miRNA target prediction in near-real time. We select to focus on DIANA microT method, since it is among the most accurate and popular ones. We study its involved sequence alignment process because it is computationally intensive. This process involves a novel type of sequence alignment, thus, we formalise this query type by introducing the ARSM problem. Moreover, since the state-of-the-art algorithms fail to perform well for the aforementioned queries, we propose a new algorithm, termed PS-ARSM, which takes advantage of special characteristics of these queries to avoid redundant computations. The methods discussed and the results obtained appear in [98].

2. Since miRNA target prediction methods also incorporate some other computational intensive processes, except sequence alignment, we investigate the option to distribute these processes in the nodes of a Cloud infrastructure. Therefore, we designed two Cloud-based target prediction systems, termed TarCloud and MR-microT. The former was developed using the framework of Microsoft Azure, while the latter is a MapReduce implementation using the Hadoop framework. Our measurements show that both systems accelerate the prediction process, with MR-microT being superior in many aspects. The methods discussed and the results obtained can be found in [97] and [40].

3. Regarding the dissemination of miRNA related information, we performed substantial work to provide valuable tools to the researchers of the field. In particular, working together with the team of Prof. Artemis Hatzigeorgiou at BSRC “Al. Fleming”, we collected data scattered to many scientific publications and databases, combined them and processed them to extract useful knowledge. The results are distributed to the research community through a multitude of powerful tools having intuitive Web interfaces. In particular, we developed (a) DIANA microT, which provides to life scientists predictions for the genes that are targeted by all the known miRNAs, (b) DIANA miRGen, that informs its users about the genomic locations of all miRNA transcripts and their expression behaviour, (c) DIANA TarBase, that provides experimentally verified miRNA targets, (d) DIANA mirPath, which investigates the role of miRNAs in the known metabolic pathways, and (e) DIANA mirPub, a tool assisting life scientists in miRNA-related literature search. The aforementioned systems and the methods discussed appear in [59, 60, 76], [3], [99], and [100].

4. During the development of DIANA TarBase we recognised the difficulties that curators of scientific databases face when they need to identify miRNA-gene interactions recorded in the text of relevant publications. This was the motivation to investigate the opportunities in automatic recognition of miRNA-gene interactions. The results of the preliminary evaluation, presented in [95], make us hopeful about providing adequate suggestions to DIANA TarBase curators.

5. Finally, we introduced Hitmap, an indexing approach that supports efficient alignment for long DNA reads lengths and relatively large error thresholds. Hitmap fills the gap in DNA read alignment approaches since it outperforms the state-of-the-art in the case of long DNA reads while its performance for
short read alignment remains similar to the best of short DNA read alignment algorithms.

1.2 Outline

The remainder of this thesis is structured as follows.

Chapter 2 establishes the necessary background for introducing our proposed methodology. In particular, it discusses the most popular problems in data management for sequence databases and introduces their state-of-the-art solutions. Moreover, it introduces the reader to some necessary concepts from biology and discusses the data management and processing needs that emerge in life sciences.

In Chapter 3 we discuss our efforts to boost miRNA target prediction. First, we study the sequence matching process that consists the first step of DIANA microT and we propose an approach to accelerate this process. Since target prediction methods also incorporate other computational intensive processes, besides sequence matching, we develop two Cloud-based approaches that distribute these processes in many computational nodes. Finally, in Chapter 3.3 we sum up the work we have done in this field and we discuss our contribution.

In Chapter 4, we present a set of infrastructures developed to support miRNA research. Until recently, important information about the function and the regulation of each miRNA was scattered in many databases or even not available at all. This was an important obstacle for researchers in life sciences who were trying to understand the role of miRNAs in many biological pathways, a knowledge that could help towards discovering treatments for particular diseases. We present in detail one by one all the tools we developed to assist miRNA research discussing their motivation, their functionality and their contributions.

In Chapter 5, we present Hitmap, an index structure supporting alignment for large read lengths and edit distance thresholds.

Chapter 6 concludes the discussion of this thesis summarizing its contributions. Finally, we identify possible extensions and propose future work.
Chapter 2

Preliminaries

In this chapter we provide the necessary background for understanding the problems and the methods introduced in the next chapters. In particular, Chapter 2.1 establishes some preliminary notions regarding sequence management and presents the most important sequence matching problems along with their state-of-the-art solutions. Chapter 2.2 introduces the reader to some necessary concepts from biology and discusses the data management and processing needs that emerge in life sciences.

2.1 Management of sequence databases

In many fields (like computational biology, signal processing, text retrieval, etc) researchers collect data represented as ordered sets of symbols, called sequences, to extract useful information from them by performing several types of analysis (e.g., search for common patterns that appear inside them). The symbols in these sequences belong to a finite set, called alphabet. The size of the alphabet and its contents depend on the application. For instance, a life scientist may use a sequencing machine to identify the sequence of nucleotides that controls the production of a particular protein in a cell. These sequences of nucleotides are called genes and their alphabet contains 4 symbols (since there are 4 distinct types of nucleotides).

For the rest of this thesis, we use capital Latin letters, like $S$, to represent sequences. We use the letter $\Sigma$ to represent the alphabet of our sequences, i.e., $\forall S \in \Sigma$. $|S|$ denotes the length of $S$, i.e., the number of symbols it contains. For any $i, j \in [1, |S|]$, $S[i]$ corresponds to the $i$-th symbol in $S$, while $S[i, j]$ to the subsequence of $S$ that starts at the $i$-th and ends at the $j$-th symbol. We use the notation $S[i, j] \subseteq S$ to indicate that $S[i, j]$ is a subsequence of $S$.

A sequence database is a database that stores a multitude of sequences along with some metadata that annotate them. The most common queries on sequence databases are selection queries based on the similarity of a given query sequence to any of the data sequences or to any part of them. This evaluation of such queries is usually referred as sequence matching or sequence alignment or, simply, sequence searching. In Chapter 2.1.1 we describe the most popular sequence matching problems, while in Chapter 2.1.1.3 we discuss the state-of-the-art algorithms and index structures that deal with each of them.
2.1.1 Sequence matching problems

2.1.1.1 Exact sequence matching

The simplest selection query that can be posed to a sequence database is the one that requests to retrieve all the database records that contain an exact occurrence of a given query sequence. This problem is known as the exact sequence matching problem (ESM) and the selection queries of this type are called ESM queries. For example, consider a sequence database \{‘girgoa’, ‘roge’, ‘trvtt’\} and an ESM query ‘roge’. The answer to this query is the set of the first two records of the database.

There are many known efficient algorithms that can be used for ESM query evaluation. The most popular is the Boyer-Moore algorithm [11]. The algorithm preprocesses the query sequence and uses information gathered during preprocessing to skip sections of the database sequences. The algorithm has worst-case execution time of \(O(n + m)\), where \(m\) is the length of the query sequence and \(n\) is the sum of the lengths of the database sequences. Note that Boyer-Moore algorithm, combined with some other techniques, is “under the hood” of the popular GREP tool that is provided by Unix-based systems to search inside files.

Moreover, there are many index structures that can be used to boost the ESM query evaluation. The most popular are the suffix tree [102] and the suffix array [57]. Both are based on the same idea and belong to the family of suffix indices. Given a sequence \(S\), its suffix tree is a tree that contains one leaf for each suffix of \(S\). Each edge of the tree is labeled with a subsequence of \(S\). Any path from the root to a leaf encodes a suffix of \(S\) (the suffix of the path can be found by concatenating the labels of the edges in the order of browsing). In the case of a database of sequences, one tree that contains the suffixes of all the database sequences can be constructed. Trees like the latter are known as generalised suffix trees.

After building a suffix tree on a sequence \(S\), ESM queries for query sequences of length \(m\) can be answered in \(O(m)\) average time. This is done by starting a traversal from the root node of the tree, while reading symbols of the query sequence (from the left to the right). During this procedure, the next edge of the tree to be traversed is selected based on the next symbols to be read from the query sequence. In particular, among the outgoing edges of the current node, the one having prefix that matches exactly with the next query sequence symbol is selected for traversal. If there are unmatched symbols at the end of the selected node or if there is no such edge, the traversal stops. The answer to the ESM query can be found by processing the leaves contained in the subtree defined by the position in the tree where the traversal ended.

One major drawback of the suffix tree is that it has very large size (it is some times larger than the initial sequence). A suffix array is a compressed version of suffix tree. Any algorithm that utilises a suffix tree can be adapted to work with suffix arrays with some additional cost in execution time. For instance, the evaluation of ESM queries can be performed in \(O(m + \log n)\) time with a suffix array. By enhancing some additional information to the suffix array, the same complexity of execution time as the one of suffix trees can be achieved [1].

In general, ESM is a well-studied problem and many satisfying solutions exist for almost any of its variations. More challenging are selection queries that request to retrieve database records that contain approximate occurrences of a given query sequence. Our interest is focused on such queries, thus, in Chapter 2.1.1.3 we are
not going to discuss any ESM solutions. However, the reader can find more details about ESM algorithms in [26].

2.1.1.2 Approximate sequence matching

In the heart of most sequence matching problems there is a selection query on a sequence database, where the matching criterion is based on sequence similarity. This broad family of problems is known as approximate sequence matching (ASM) or sequence alignment problems. An excellent review on these problems can be found in [27].

There are many measures of sequence similarity. The most popular and powerful of them is the edit distance[50, 51]. Given two sequences, $S_1$ and $S_2$, their edit distance $ed(S_1, S_2)$ is defined as the minimum number of symbol insertions, deletions, and replacements that can be performed to transform the one sequence to the other. For example, it holds that $ed(TOP, TAP) = 1$, since we can transform $TOP$ to $TAP$ by replacing the symbol $O$ with an $A$. In fact, edit distance captures how dissimilar two sequences are, thus, small edit distance is translated in high similarity. Note that edit distance has the properties of a metric.

Another commonly used measure of similarity is Hamming distance. Given sequences $S_1$ and $S_2$, their Hamming distance $hd(S_1, S_2)$ is defined as the minimum number of symbol replacements that can be performed to transform the one sequence to the other. Hamming distance is also a metric; in fact, it is a special case of the edit distance where only symbol replacements are allowed.

There are many other similarity measures (such as Smith-Waterman similarity [90]), however, for now on, we consider that edit distance is used in all cases, unless stated differently.

There are two main categories of approximate sequence matching problems. The one incorporates the retrieval of the $k$ most similar database sequences to a given query (top-$k$ sequences). The other consists of retrieving all the database sequences having edit distance from the query at most $k$, where $k$ is a threshold given by the user. In both categories, the retrieved sequences, called query matches or alignments, can be either entire records of the sequence database (global alignment [71, 84]) or subsequences of them (local alignment), depending on the objective of the problem. Since studying top-$k$ queries is out of the scope of the present dissertation, the interested reader could refer to [27].

Regarding the local alignment queries, there is the option to search for alignments of parts of the query [91]. In this case, for each database record, the objective is to find the pair of query and record subsequences that achieve the most similarity and then examine if their similarity satisfies the given threshold $k$. We refer to these queries as double-local alignment queries.

2.1.1.3 Algorithms and index structures for ASM

Dynamic programming approaches, like the one presented in [84], are usually used to evaluate global alignment queries. The basic dynamic programming approach requires $O(s_1 \cdot s_2)$ time to calculate the edit distance of two sequences $S_1$ and $S_2$, where $s_1$ and $s_2$, in respect, are their lengths.

Regarding the local alignment queries, there exist several works surveying related algorithms [36, 67]. The basic solution to find the alignments of a given query
Q inside a database record D, called the “data sequence”, is again a dynamic programming approach [85]. Its running time is \( O(q \cdot d) \), where \( q \) and \( d \) are the lengths of \( Q \) and \( D \), respectively. Several improvements to this basic dynamic programming approach have been proposed (for details refer to [67]). One such optimisation, achieving \( O(k \cdot d) \) running time, where \( k \) is the edit distance threshold, is the cut-off heuristic [96]. Its basic idea is to avoid calculating parts of the dynamic programming table based on a heuristic that identifies table cells for which it is sure that their value will be greater than \( k \).

However, the most efficient algorithms for local alignment queries belong to the group of filtering algorithms. In particular, Chang’s [15] and Fredriksson’s [22] algorithms have optimal average case time complexity. These algorithms first compare the query to any possible sequence of a predetermined length \( \ell \), called \( \ell \)-gram, and then use this information to filter out areas of the data sequence that cannot contain any alignment. The remaining areas are processed using a conventional local alignment algorithm.

Interesting issues arise when a large number of queries must be evaluated in the same database. If the number of queries is large, then it is sure that there are overlaps between them. The independent evaluation of queries that have overlaps results in poor performance due to redundant computations. This was the motivation for the introduction of multiple local alignment algorithms (or, in general, multiple approximate sequence matching algorithms). Several methods exist (e.g., [66, 9, 34, 22]), however, Fredriksson’s algorithm [22] has been proven to be optimal [70]. This algorithm scans each data sequence using a sliding window. For each window position, it reads backwards (i.e., from the right to the left) consecutive, non-overlapping \( \ell \)-grams. When the aggregated deviation of the read \( \ell \)-grams exceeds a threshold, the algorithm skips the current window and slides it to the right. Otherwise, it must examine the window for results. Note that these methods are designed for queries of similar length, otherwise it is not possible to find a proper \( \ell \) value that achieves good performance for all of them.

In what it concerns double-local alignment queries, the dynamic programming of Smith-Waterman [91] guarantees to solve this problem in \( O(q \cdot d) \) time, for an important class of similarity measures. When searching for highly similar subsequences, the high computational cost of dynamic programming algorithms makes approximation solutions (e.g., [56, 77, 4, 5]) more attractive. These methods use heuristics to avoid searching parts of the sequences that are unlikely to contain a double-local alignment. As a side-effect they may miss results. The most known approximation solution is BLAST [4] and its variations [5, 107, 45, 42].

Several index structures can be applied to boost the ASM problems. First of all, there are algorithms utilising ESM indices, like suffix trees and suffix arrays, to evaluate ASM queries [69]. This is possible since, usually, an ASM query can be translated into a set of ESM queries. This central idea is also in the core of the most popular family of ASM index structures, the gram-based inverted indices. These structures are just hash-based dictionaries containing one entry for each distinct subsequence of the data having a predefined length. Each entry has as key the sequence and as value the list of all the positions inside the data where it appears. The evaluation of local alignment queries, under an edit distance threshold \( k \), can be performed using a gram-based inverted index based on a theorem known as the pigeonhole principle.
The original form of the pigeonhole principle was introduced by P.G.L. Dirichlet in the context of discrete mathematics. Its adaptation used by gram-based inverted indices states that if we divide a query $Q$ in $k + x$ consecutive, non-overlapping fragments\(^1\), then any alignment of $Q$ in $S$ will contain exact appearances of at least $x$ fragments. If the condition of this principle does not hold for a given data subsequence, then it is not possible that this subsequence contains a $Q$ alignment.

Based on the above discussion, consider a query $Q$, an edit distance threshold $k$ and a gram-based inverted index built on the data $D$ using $\ell$-grams, where $\ell = \lceil \frac{q k}{k + x} \rceil$. Then, $Q$ can be divided in $k + x$ consecutive, non-overlapping fragments and the inverted lists of all these fragments can be retrieved from the index. The only $D$ positions that may contain $Q$ alignments are those appearing for at least $x$ times in the retrieved lists. To find them a special merging of the lists must be applied. The aforementioned method has been introduced in [68], while some optimisations regarding the final list merging step have been proposed in [52].

Since the local alignment problem became viral in Bioinformatics due to its relation with the DNA read alignment process (see Chapter 2.2.3) many hash-based approaches including Maq [29], SOAP [80], GSNAP [103] were introduced by researchers of the field. These approaches reduce the problem of local alignment to ESM of fragments of the reads, however most of them use arbitrary query fragmentation, not fragmentation based on the pigeonhole principle. Approaches that outperform the previous, using compressed suffix arrays appeared also in Bioinformatics (e.g., BWA [53], Bowtie [10] and SOAP2 [81]). These methods use Burrows-Wheeler Transforms (BWT) technique [13] to compress the arrays and achieve very small memory footprint.

However the compressed suffix array methods are optimised for extremely small query lengths (e.g., queries of $30 - 50$ symbols). Moreover, they perform well when they are used for matching with a small number of errors [54]. However, technological advances in the equipment producing DNA reads raise the need for aligning approaches that support local alignment for longer reads and more tolerant to alignment errors. Furthermore, since the cost of computer memory continues to decrease, there is no need to keep a small memory footprint. The previous arguments became the motivation of a new gram-based inverted index approach, called WHAM [54].

The central idea of WHAM is the same as the one of the basic gram-based inverted index [68]. However, each WHAM entry, instead of capturing the appearances of any possible query fragment in the data, it captures the combined appearances of possible fragments. The number of fragments in each combination is based on the pigeonhole principle. In particular, if a query $Q$ must be aligned with at most $k$ edit distance and we select to split the query in $k + x$ fragments, then each WHAM entry captures any combined appearance of $x$ fragments in $D$. In this way, a local alignment query can be answered simply by taking the union of the inverted lists of all query fragment combinations. The only aspect that must be done with attention is to consider possible insertions and deletions in each fragment combination by shifting each fragment with any possible way. Note that WHAM becomes identical to the basic gram-based inverted index in case that $x = 1$.

Finally, except of gram-based and suffix arrays-based solutions, there is also RBSA [75] a reference-based approach. RBSA is based on the metric property of edit distance. In particular, for each data subsequence its edit distance to a set

\(^1\)Note that there may exist symbols at the end of $Q$ that do not belong to any fragment.
of reference sequences is calculated. During query evaluation, the edit distance of the query to the same reference sequences is also computed. Consider a random subsequence $A$ of the data and let $Q$ be the query and $k$ the edit distance threshold. If there exists a reference sequence of $A$, say $R$, for which $|ed(A, R) - ed(Q, R)| > k$, then, $ed(A, Q)$ cannot be smaller or equal to $k$. Examining the aforementioned condition for all the reference sequences of any $D$ subsequence is less computational demanding than calculating the edit distance of $Q$ to the subsequence. Therefore, RBSA provides a way to filter out areas of data.

2.2 Data management and processing in Life Sciences

The life sciences comprise the fields of science that involve the scientific study of living organisms and life mechanisms, in general. Two types of biological molecules play the most important role in these mechanisms: nucleic acids and proteins. Nucleic acids are chains of monomers called nucleotides, while proteins are chains of amino acids. Many life mechanisms incorporate interactions between the aforementioned biological molecules based on the sequence of compounds in their chains. This is the reason why representing nucleic acids and proteins as sequences and using sequence matching algorithms to analyse them ignited the development of new fields, such as bioinformatics and computational biology. Research in these fields involves finding efficient ways to manage and process biological sequences.

In Chapter 2.2.1 we are going to introduce the reader to some necessary concepts from biology. In Chapters 2.2.2 and 2.2.3 we are going to discuss miRNA target prediction and DNA read alignment, respectively, which are the two problems from the field of bioinformatics that motivated our work.

2.2.1 Concepts from biology

Nucleic acids are large biological molecules (or macromolecules), essential for all known forms of life. They are made of monomers known as nucleotides. Each nucleotide has three components: a 5-carbon sugar, a phosphate group, and a nitrogenous base. If the sugar is deoxyribose, the polymer is DNA. If the sugar is ribose, the polymer is RNA.

The two basic nucleic acids are DNA (deoxyribonucleic acid) and RNA (ribonucleic acid). These nucleic acids, together with proteins, are the most important biological macromolecules and can be found in abundance in all living organisms. Their function is to encode, transmit, and express genetic information. Below we elaborate on the functions of the most important types of these biomolecules.

2.2.1.1 DNA, genome, and genes

DNA is a molecule that encodes the genetic instructions used in the development and functioning of all known living organisms and many viruses. Most DNA molecules consist of two biopolymer strands coiled around each other to form a double helix. The two DNA strands are known as polynucleotides since they are composed of simpler units called nucleotides.
Each nucleotide is composed of a nitrogen-containing nucleobase as well as a monosaccharide sugar called deoxyribose and a phosphate group. There are four types of nucleobases: guanine (G), adenine (A), thymine (T), or cytosine (C). The nucleotides are joined to one another in a chain by covalent bonds between the sugar of one nucleotide and the phosphate of the next, resulting in an alternating sugar-phosphate backbone. According to base pairing rules (A with T and C with G), hydrogen bonds bind the nitrogenous bases of the two separate polynucleotide strands to make double-stranded DNA.

Within cells, DNA is organized into long structures called chromosomes. During cell division these chromosomes are duplicated in the process of DNA replication, providing each cell its own complete set of chromosomes. Eukaryotic organisms (like animals and plants) store most of their DNA inside the cell nucleus and some of their DNA in organelles, such as mitochondria or chloroplasts. In contrast, prokaryotes (bacteria and archaea) store their DNA only in the cytoplasm. For now on, our focus will be on the eukariotic organisms.

The set of all DNA molecules contained in any cell of a eukariotic organism is called the genome. The genome is the exactly the same for all the cells of an individual	extsuperscript{2}. It resides inside a protected area of the cell, called nucleus. As mentioned previously, the genome is organised into chromosomes. Inside the DNA chains of the chromosomes there are some regions, called genes, each of which encodes the sequence of amino-acids that forms a protein. Proteins are the most important macromolecules of the cell since they perform a vast array of important functions including catalyzing metabolic reactions, replicating DNA, responding to stimuli, and transporting molecules from one location to another. This is why genes attracted significant focus by researchers.

The rest of the genome is known as the non-coding genome and for decades scientists were believing that it is just a biproduct of evolution. However, during the last decades life scientists found that these areas are also functional (see also Chapter 2.2.1.3).

2.2.1.2 Protein synthesis and messenger RNA

Proteins are assembled from amino acids using information encoded in genes. Each protein has its own unique amino acid sequence that is specified by the nucleotide sequence of the gene encoding this protein. The genetic code is a set of three-nucleotide sets called codons and each three-nucleotide combination designates an amino acid, for example AUG is the code for methionine. Because DNA contains four different nucleotides, the total number of possible codons is 64. Therefore, there is some redundancy in the genetic code, with some amino acids specified by more than one codon.

Genes are first transcribed into pre-messenger RNA (pre-mRNA) by proteins, such as RNA polymerase. Most organisms then process the pre-mRNA (also known as a primary transcript) to form the mature mRNA or mature transcript, which is then used as a template for protein synthesis. Eukaryotic organisms make mRNA in the cell nucleus (where the genome resides) and then translocate it across the nuclear membrane into the cytoplasm, where protein synthesis then takes place.

The process of synthesizing a protein from an mRNA template is known as

\textsuperscript{2}There are some exceptions, such as the gametes.
The mRNA is loaded onto a large and complex molecular machine, known as ribosome. Then, it is read three nucleotides at a time by matching each codon to its base pairing anticodon located on a transfer RNA molecule, which carries the amino acid corresponding to the codon it recognizes. During the process the amino acids are placed together to form large chains, which are the proteins (or parts of the proteins).

Since genes contain all the information used by the cell to synthesise a protein, when the protein encoded by a gene is produced, we say that this gene is expressed.

2.2.1.3 MicroRNAs

MicroRNAs (miRNAs) are single-stranded RNA molecules of approximately 20–30 nucleotides which do not code any protein. Instead, they function as regulators of gene expression by binding to messenger RNA (mRNA) molecules and destabilizing them or inhibiting their translation. MiRNAs are found to be implicated in a wide range of physiological molecular processes, and their deregulation leads to diverse diseases [23, 21, 49].

MiRNAs are located in intergenic regions or in the introns of protein coding genes. They are transcribed by RNA Polymerase II as independent transcripts or as part of the transcript of a host gene. Only a small group of miRNAs located inside ALU repetitive elements is transcribed by RNA Polymerase III. A miRNA transcript can host more than one miRNA and can be several thousand nucleotides long including introns.

A promoter region is located around the transcription start site (TSS) of a transcript and is regulated by proteins that bind to this region. Evidence thus far suggests that binding sites for transcription factors (TFs) are similarly distributed within the promoters of both protein coding genes and miRNA transcripts [62]. MiRNA primary transcripts (pri-miRNA) are processed in the nucleus to form pre-miRNAs, approximately 70-nucleotide stem-loop structures also called miRNA hairpins. These are later processed into mature miRNAs in the cytoplasm via interaction with the endonuclease Dicer, which also initiates the formation of the RNA-induced silencing complex (RISC). Since primary transcripts are short lived and present only inside the nucleus, it is hard to identify them with standard molecular techniques.

After the Dicer enzyme cleaves the pre-miRNA stem-loop, two complementary short RNA molecules are formed, but only one of them, the guiding strand, is predominantly integrated into the RISC complex. The remaining strand, known as the miRNA*, anti-guide or passenger strand, is usually degraded. However, the proportion of the integration of each strand varies with the miRNA species, with some miRNAs having almost equal abundance of each of the two strands incorporated into RISC. Another common nomenclature for complementary miRNA strands is the ‘-3p’ and ‘-5p’ naming convention. These names do not imply which miRNA is more commonly incorporated to the RISC complex. The miRNA-miRNA* and miRNA-3p-miRNA-5p nomenclatures are both widely used in the community, often to denote the same complementary miRNA pair. Mature miRNA molecules are bound by the RISC complex, are guided to specific motifs within the 3’-UTR of protein coding mRNAs, and prevent these mRNAs from being translated into protein. The biogenesis of miRNAs and their regulation by TFs is diagrammed in Figure 2.1.
2.2.2 Computational methods for microRNA target prediction

Knowing the miRNAs that target a particular gene helps to understand the causes of human diseases and searching for treatments. However, the biochemical experiments that reveal miRNA targets are costly and time-consuming. Therefore, a multitude of computational methods to predict miRNA targets, called miRNA target prediction methods have been proposed. The first such methods were developed back in 2003. During the last decade, more than a dozen of such methods were proposed, making the field of miRNA target prediction one of the most active in bioinformatics. An excellent survey of the field can be found in [87, 2].

One of the most accurate and popular miRNA target prediction methods is DIANA microT. Since in this thesis we present many efficient algorithms and systems based on this method (see Chapter 3), in the next chapters we discuss the most recent versions of this method.

2.2.2.1 DIANA microT v.3

In the base of DIANA microT v.3 prediction method [59] is a set of biological observations. A typical miRNA is approximately 20 – 30 nucleotides long, but the nucleotides close to its 5’-end are crucial for recognizing a target sequence and binding to it. Usually, a strong binding (i.e., at least 7 consecutive Watson-Crick base pairing nucleotides) between the first 9 nucleotides from the 5’-end of the miRNA sequence (here called the miRNA driver sequence) and the target gene is required for sufficient repression of protein production. However, there is experimental evidence [12] that a weaker binding, involving only 6 consecutively paired nucleotides or including G:U wobble pairs, can also repress protein production if there is additional binding between the miRNA 3’-end and the target gene.

The DIANA microT v.3 method considers as candidate binding sites, those UTR sites that have consecutive Watson-Crick base pairing of 7, 8, or 9 nucleotides with the miRNA, starting from the first or the second position from the 5’-end of the miRNA. For sites with additional base pairing involving the 3’-end of the miRNA, a single G:U wobble pair or binding of only 6 consecutive nucleotides to the driver
sequence are also allowed. Using as features the binding type and the conservation profile of the candidate binding sites, a score is calculated for each of them through comparative analysis versus a set of candidate binding sites identified based on mock miRNA sequences. The overall miRNA target prediction (miTG) score is calculated as the weighted sum of the scores of all identified candidate binding sites on the 3’-UTR. The method uses up to 27 species to assess the candidate binding sites conservation profiles taking into account both conserved and non-conserved candidate binding sites for the estimation of the final miTG score.

For the evaluation of each miRNA’s predicted interactions, the method compares them to those predicted for a set of mock miRNAs. Mock miRNAs are independently created for each real miRNA and are designed to have approximately the same number of predicted targets as the real miRNA. This allows the calculation of miRNA-specific SNR at different miTG score cut-offs as well as the estimation of a precision score that provides an indication of the false positive rate of a particular miTG interaction.

2.2.2.2 DIANA microT v.4

DIANA microT v.4 target prediction method [60] is almost the same as DIANA microT v.3 method. The only improvement is that, unlike the previous version that was based on features separating real and mock (shuffled) miRNAs (see Chapter 2.2.2.1, DIANA microT v.4 uses high throughput experimental data for the same purpose. These data were obtained by the experiments of Selbach et al. in [83].

2.2.2.3 DIANA microT v.5

DIANA microT v.5 method [78] is based on the previous versions of DIANA microT and on features extracted from existing mammalian high-throughput immunoprecipitation and sequencing data. The analysis is performed independently for the coding and 3’-UTR regions of genes and reveals different sets of features and models for the two regions. The two models are combined into a novel computational model for miRNA target genes, which achieves higher sensitivity compared with other popular programs and the previous versions of DIANA microT. Further analysis indicates that genes with shorter 3’-UTRs are preferentially targeted in the coding regions, suggesting that evolutionary selection might favor additional sites on the coding regions in cases where there is restricted space on the 3’-UTR. Note that, in fact, the 3’-UTR model is the same as the previous DIANA microT methods.

2.2.3 DNA read alignment

Studying mechanisms of life is performed mainly by conducting biochemical experiments that investigate possible interactions arising between biomolecules (like DNA, RNA, and proteins). However, such experiments are costly and time consuming. Therefore, investigating the possible interaction between two molecules which are not related is just a waste of time and money. This is why researchers represent these molecules as symbol sequences and apply computational processing approaches to reveal possible interactions.

The computer representations of biomolecules are produced using machines of special purpose, called sequencing machines. These machines get biological samples
containing cells from particular tissues and produce small sequences, each of which represent a part of the DNA contained to the sample. The aforementioned small sequences are called DNA reads, or simply reads. Multiple reads extracted by samples of the same living organism can be aligned to each other to produce the whole genome sequence of this organism. Moreover, many other important types of DNA analysis consist of aligning DNA reads in an already known genome sequence. The term DNA read alignment is used to describe all the previous types of analysis.

At its heart, the read alignment problem is similar to the common sequence alignment problem in data processing (see Chapter 2.1.1.2). Therefore, all methods described in Chapter 2.1.1.3 can also be applied to produce the alignments of DNA reads.

The first automated sequencing machine was introduced by Applied Biosystems in 1987 and enabled the completion of the human genome project in 2001. It used the Sanger sequencing method, a technology used by all the first generation sequencing machines.

The human genome project motivated the development of a new generation of cheaper, high-throughput sequencing machines. This generation, known as next generation sequencing machines, includes the 454, SoLid and Illumina DNA sequencing platforms. Next generation sequencing machines have boosted the rate of DNA sequencing in comparison to the previous Sanger-based methods. The DNA reads produced by these machines are very small consisting of 30 – 50 symbols. They may also contain errors caused by DNA sequence technique or by a process applied after it, PCR amplification. In the last decades, many sequence alignment methods have been proposed and have been proven to work adequately for DNA reads having the aforementioned characteristics (see Chapter 3.1.4.2).

However, small read sizes can result in problems during the sequence processing. For example, the alignment of small DNA reads can be dramatically affected by the presence of structural variations or single-nucleotide polymorphisms (SNPs) inside them. This motivated a new generation of sequencing machines that can produce much longer reads. For instance, Ion Torrent PGM produces reads of 200 – 400 symbols, 454 GS FLX reads of 700 symbols, and PacBio reads containing several thousands of symbols. A side effect is that, in order to provide this benefit, these machines usually sacrifice accuracy, which means that their reads contain increased number of misplaced symbols. This creates a challenge in sequence alignment since state-of-the-art approaches are optimised for small reads with small number of errors and their performance is poor for long and not accurate reads.
Chapter 3

Efficient miRNA target prediction

For many years, life science researchers used to consider that only the regions of genome which translate into proteins are important for life. This has dramatically changed after the discovery, in the late 1990s, of regions in the “non-translated” genome playing a key role in several life functions. Among those functions, one of the most important is the deactivation (“silencing”) of genes by small RNA molecules, called microRNAs (miRNAs). In brief, each miRNA targets particular genes, destroying their transcripts and, consequently, prohibiting the production of the encoded protein (see Chapter 2.2.1).

Knowing the miRNAs that target a particular gene helps to understand the causes of human diseases and to develop treatments for them. However, biochemical experiments that can determine targets are costly and time-consuming. Therefore, computational methods to predict miRNA targets have been proposed (see Chapter 2.2.2). Although these methods boost biochemical experiments, they contain heavy computations and, thus, they require considerable time to produce the target of all known miRNA sequences.

We selected to study and boost DIANA microT (Chapter 2.2.2.3) one of the most popular and accurate miRNA target prediction methods. In this chapter we consider our work in this direction. First, in Chapter 3.1 we study the sequence matching process that consists the first step of DIANA microT and we propose an approach to accelerate this process. Since target prediction methods also incorporate other computational intensive processes, besides sequence matching, in Chapter 3.2 we develop two Cloud-based approaches that distribute these processes in many computational nodes. Finally, in Chapter 3.3 we sum up the work we have done in this field and we discuss our contribution.

3.1 Efficient sequence matching for target prediction

3.1.1 Motivation and contribution

Sequence matching problems (e.g., exact/approximate, local/global alignment) have been extensively studied, and several algorithms, reviewed in [82, 26], have been proposed (see Chapter 2.1.1.3). These problems are very popular as they naturally appear at the heart of many diverse applications. For example, in biological databases,
which contain long sequences of symbols (such as nucleotides, amino acids, etc.), sequence matching algorithms help identify homologous (i.e., of similar functionality) biological entities, such as genes, proteins, etc (see also Chapter 2.2).

Often, advances in particular research fields introduce complex matching criteria that give rise to novel sequence matching problems. For example, it has been observed that a chemical association, known as binding, of a non-coding RNA sequence (e.g., miRNA, short interfering RNA, etc.), termed the pattern, with a larger one (e.g., a gene), termed the data, usually occurs around a key location of the pattern, called the core (e.g., the nucleotides near the start of the micro-RNA [19]). Since laboratory experiments are costly and time-consuming, computational methods to predict bindings based on the previous observation have been proposed (see Chapter 2.2.2). For instance, the prediction algorithm DIANA microT (see Chapter 2.2.2.1) employs a conventional sequence matching algorithm to test whether any super-sequence of the core, termed region, matches approximately (i.e., a few individual symbols may mismatch) with a subsequence of the data. This process is repeated for each region, and the maximum number of mismatches allowed is set empirically based on the region’s length. The larger a region is and the better it matches with the data, the more likely a binding is.

Motivated by this real-life case, we generalized the above matching criteria and introduced the Approximate Regional Sequence Matching (ARSM) problem [98] (see also Chapter 3.1.2. Assume a data sequence $S$, a pattern sequence $P$, and a core (i.e., a subsequence) of $P$. Briefly, an ARSM result is a subsequence of $S$ that approximately matches some subsequence of $P$ under the following conditions: (a) the $P$ subsequence is a region, i.e., it encloses the core, and (b) the allowable deviation, in terms of the number of mismatching symbols, between the subsequence of $S$ and the $P$ region grows with the latter’s length.

Figure 3.1 presents an ARSM instance. The top part of the figure depicts the data sequence $S$, while the bottom shows three copies of the pattern sequence $P$ aligned in different locations under $S$. The dark shaded part in each $P$ copy corresponds to the core region. On the other hand, the light shaded part depicts a $P$ subsequence (different in each copy) that matches with the corresponding light shaded subsequence of $S$. In each $P$ subsequence, the number underneath it denotes its length, while a cross indicates a mismatching symbol with respect to $S$. Furthermore, the allowed number of mismatching symbols is 1 (resp. 2) for regions of length 8 (resp. 10).

Observe that the $S$ subsequence that matches with the second subsequence of $P$ is not an ARSM result because this $P$ subsequence has length 8 and contains more mismatches than allowed. Moreover, neither the $S$ subsequence corresponding to the third subsequence of $P$ is an ARSM result, since the $P$ subsequence does not enclose the core, i.e., it is not a region. On the other hand, the $S$ subsequence
corresponding to the first $P$ subsequence is an ARSM result as it satisfies both conditions.

The distinctive characteristic of ARSM, compared to other approximate sequence matching problems, is that multiple sequences, the regions, are examined for matches under varying allowable deviation values. Note that it is possible to extend existing methods to solve the ARSM problem. The naïve approach is to apply a state-of-the-art approximate sequence matching (ASM) algorithm (e.g., [67]) for every possible region. Clearly, this brute-force method is inefficient as it makes no effort to share computation among regions that are overlapping.

A better alternative is to apply a multiple ASM (MASM) algorithm (e.g., [66, 22]) that is able to process multiple patterns at a time and exploit their overlaps. Since MASM algorithms are designed to operate on a set of patterns of equal length (see also Chapter 2.1.1.3), a MASM-based approach must first group regions according to their length, and execute MASM once per group. However, this method cannot take advantage of the overlaps in regions across groups. Besides, for genomic databases, i.e., with small alphabet size (4 symbols), short patterns (a few dozens of symbols), and large allowable deviations (around 20% of the pattern length), MASM algorithms are known to suffer [22].

Note that local alignment algorithms (e.g., Smith-Waterman algorithm [91]), which search for matches of all possible pattern subsequences (and thus of the regions as well), cannot be adapted to the ARSM problem for three reasons. First, they require that the allowable deviation is fixed and independent of the subsequence length. Second, the popular state-of-the-art heuristic algorithms (such as BLAST [4]) do not identify all matches. Third, and more importantly, even if exact algorithms are used, some ARSM answers may still be missed. Consider, for example, the data sequence $S = \cdots GTTGA \cdots$, and the region $R = GCCGA$. Clearly, there exists an occurrence of $R$ in $S$ with cost (edit distance) 2. However, SW would fail to identify it\(^\dagger\). The reason is the following. In the DP array, the cell corresponding to the two $A$s in the sequences has the highest value 2, meaning that there exist two subsequences ending in $A$ that have similarity score 2. However, since this score corresponds to the $GA$ subsequences, there is no way to backtrack and identify the $GCCGA$, $GTTGA$ subsequences.

To overcome the previous limitations, we propose the PS-ARSM method [98], which takes advantage of the prefix and suffix overlaps among the regions (see also Chapter 3.1.4). Briefly, our method first determines the data subsequences where the smallest region (the core) matches under the largest possible allowable deviation. Then, based on a set of sound and complete expansion rules, the algorithm progressively expands the data subsequences to derive all ARSM results.

In the next chapters we introduce the ARSM problem, we describe the PS-ARSM method and we validate its efficiency compared to ASM and MASM based approaches by performing experiments on genomic databases.

### 3.1.2 The ARSM problem

Consider an alphabet $\Sigma$. In the remainder of this chapter, each sequence $S \in \Sigma^*$. $|S|$ denotes the length of sequence $S$. $S[i]$ corresponds to the $i$-th symbol in $S$, while

\(^\dagger\)We assume that a match has score 1, whereas a mismatch, delete or insert has score $-1$. Note that a similar counter-example exists for other scores.
$S_{[i,j]}$ to the subsequence of $S$ that starts at the $i$-th and ends at the $j$-th symbol. We use the notation $S_{[i,j]} \sqsubseteq S$ to indicate that $S_{[i,j]}$ is a subsequence of $S$.

Given two sequences, we call edit transcript (or simply transcript) an ordered set $\tau$ of edit operations to transform one sequence to the other. Typically, the following edit operations are allowed for a sequence $S$:

- Insert (I) a symbol into $S$,
- Delete (D) a symbol of $S$,
- Replace (R) a symbol of $S$ with another,
- Match (M), i.e., preserve, a symbol of $S$.

The cost $c(\tau)$ of a transcript is the number of I, D, and R operations it contains.

Consider two sequences $S$ and $P$, called data sequence and pattern sequence, respectively. One can always find a transcript that transforms $P$ into any subsequence $S_{[i,j]}$ of the data sequence. We say that the pattern $P$ occurs in the data $S$ at location $[i,j]$ with transcript cost $\epsilon$, if there is a transcript $\tau$ that transforms $P$ into $S_{[i,j]}$ having cost $c(\tau) = \epsilon$. We use the notation $(P, i, j, \epsilon)$ to indicate this occurrence.

Figure 3.2 shows an occurrence $(P, 5, 12, 2)$ of pattern $P = \text{GATTACA}$ in $S = \text{AACCGAATTAGACC}$ at location $[5, 12]$ with transcript cost $\epsilon = 2$. Indeed, according to the transcript $\tau = \text{MMIMMMRM}$, $P$ can be transformed into $S_{[5,12]}$ by inserting (I) symbol A between $P_{[2]}$ and $P_{[3]}$, and replacing (R) $P_{[6]} = C$ with G; all other symbols remain unchanged (M).

Note that there can be more than one transcripts to transform $P$ in $S_{[i,j]}$ with the same cost $\epsilon$. For instance, in the example of Figure 3.2, the transcript $\tau' = \text{MIMMMRM}$ has the same cost with the depicted one.

A pattern can occur at a particular location in the data with various transcript costs. An occurrence $(P, i, j, \epsilon)$ is called minimal if it has the lowest possible cost, i.e., there exists no other occurrence $(P, i, j, \epsilon')$ such that $\epsilon' < \epsilon$. For example, the occurrence $(P, 5, 12, 2)$ in Figure 3.2 is minimal.

Given a pattern sequence $P$ and a location $[a, b]$ in $P$ termed core $(1 \leq a \leq b \leq |P|)$, a subsequence $R = P_{[i,j]}$ of $P$ is called region if $i \leq a$ and $j \geq b$. Two special regions are the core region $P_{[a,b]}$ and the pattern region $P_{[1,|P|]}$. We next introduce the ARSM problem.

**Definition 3.1 (ARSM Problem).** Given a data sequence $S$, a pattern $P$, its core $[a,b]$, and a monotonically increasing threshold function $K : \mathbb{N} \to \mathbb{N}$, the ARSM problem is to retrieve the minimal occurrences $(R, i, j, \epsilon)$ of each region $R$ of $P$, such that: (1) $\epsilon \leq K(|R|)$, and (2) no other minimal occurrence $(R', i, j, \epsilon')$, where $R \sqsubseteq R'$, has $\epsilon' \leq K(|R'|)$.
The first constraint specifies that the allowed cost for a region occurrence depends on the size of the region and is given by the threshold function. Larger regions are allowed to have larger cost. The second constraint implies that if two different regions \( R, R' \) occur at the same location \([i,j]\) of the data sequence \( S \), then only the occurrence of the largest region is returned. We call all those retrieved minimal occurrences ARSM results.

Figure 3.3 illustrates an instance of the ARSM problem in which the data sequence \( S \) has 19 symbols and the pattern \( P \) has 10. The core is the location \([5,8]\) of the pattern. In this instance, there exist 15 possible regions labelled \( R_1 \) through \( R_{15} \). Figure 3.3 also depicts the values of the threshold function \( K \) for all possible region lengths (4 up to 10).

### 3.1.3 ARSM Characteristics

Chapter 3.1.3.1 presents some key observations regarding occurrences of overlapping regions. Recall that regions are highly overlapping (see Figure 3.3), since each one is a subsequence of the pattern and a supersequence of the core. Then, Chapter 3.1.3.2 exploits these observations to introduce a set of expansion rules that construct the set of minimal occurrences of a region from those of a smaller one.

#### 3.1.3.1 Overlapping Occurrences

Consider a data sequence \( S \) and a pattern \( P \). Assuming that \( P \) occurs at location \([i, j]\) of \( S \), the next two lemmas show how this occurrence is related to an occurrence of \( P \) at locations \([i, j + 1]\) and \([i - 1, j]\). Intuitively, an edit operation \( I \) can be appended to the transcript to accommodate for the extra symbol of the data sequence at location \( j + 1 \) or \( i - 1 \).

**Lemma 3.1.** If \((P, i, j, \epsilon)\) is an occurrence of \( P \) in data sequence \( S \) and \( \tau \) is one of its transcripts, then \((P, i, j + 1, \epsilon + 1)\) is also an occurrence of \( P \) in \( S \) and \( \tau I \) is one of its transcripts.

**Proof.** The transcript \( \tau I \) contains the same edit operations as \( \tau \), and an additional \( I \) operation. Therefore, \( c(\tau I) = c(\tau) + 1 = \epsilon + 1 \). As \( \tau \) transforms \( P \) into \( S_{[i,j]} \), the last \( I \) in \( \tau I \) inserts \( S_{[j+1]} \) at the end of \( P \). Therefore, \( \tau I \), having transcript cost \( \epsilon + 1 \), transforms \( P \) into \( S_{[i,j+1]} \), i.e., \((P, i, j + 1, \epsilon + 1)\) is an occurrence of \( P \) in \( S \) and \( \tau I \) is one of its transcripts.

---

Figure 3.3: ARSM example
Lemma 3.2. If \((P, i, j, \epsilon)\) is an occurrence of \(P\) in data sequence \(S\) and \(\tau\) is one of its transcripts, then \((P, i - 1, j, \epsilon + 1)\) is also an occurrence of \(P\) in \(S\) and \(I\tau\) is one of its transcripts.

Proof. Similar to that of Lemma 3.1.

Next, consider two pattern sequences, \(P\) and \(P\gamma\), where the latter is obtained by appending symbol \(\gamma \in \Sigma\) at the end of the former. This represents the case where regions share the same prefix and differ by one symbol, e.g., regions \(R2\) and \(R4\) in Figure 3.3. Assuming \(P\) occurs at location \([i, j]\) in \(S\), the next lemma shows how this occurrence is related to occurrences of \(P\gamma\) at locations \([i, j]\) and \([i, j + 1]\). Intuitively, an edit operation (\(D\), \(R\), or \(M\)) can be appended to the transcript to accommodate for the extra symbol \(\gamma\) of the pattern sequence.

Lemma 3.3. If \((P, i, j, \epsilon)\) is an occurrence of \(P\) in data sequence \(S\) and \(\tau\) is one of its transcripts, then:

1. \((P\gamma, i, j, \epsilon + 1)\) is an occurrence of \(P\gamma\) in \(S\) and \(\tau_D\) is one of its transcripts,
2. \((P\gamma, i, j + 1, \epsilon)\) is an occurrence of \(P\gamma\) in \(S\) and \(\tau_M\) is one of its transcripts, if \(S_{[j+1]} = \gamma\),
3. \((P\gamma, i, j + 1, \epsilon + 1)\) is an occurrence of \(P\gamma\) in \(S\) and \(\tau_R\) is one of its transcripts, if \(S_{[j+1]} \neq \gamma\).

Proof. We prove case 1; cases 2 and 3 can be proven similarly. The transcript \(\tau_D\) contains the same edit operations as \(\tau\), and an additional \(D\) operation. Therefore, \(c(\tau_D) = c(\tau) + 1 = \epsilon + 1\). As \(\tau\) transforms \(P\) into \(S_{[i,j]}\) and the last \(D\) in \(\tau_D\) just deletes \(\gamma\) from \(P\gamma\), then \(\tau_D\) transforms \(P\gamma\) into \(S_{[i,j]}\). Therefore, \((P\gamma, i, j, \epsilon + 1)\) is an occurrence of \(P\gamma\) in \(S\) and \(\tau_D\) is one of its transcripts.

Finally, consider two pattern sequences, \(P\) and \(\theta P\), where the latter is obtained by appending symbol \(\theta \in \Sigma\) at the beginning of the former. This represents the case where regions share the same suffix and differ by one symbol, e.g., regions \(R1\) and \(R3\) in Figure 3.3. Assuming \(P\) occurs at location \([i, j]\) in \(S\), the next lemma shows how this occurrence is related to occurrences of \(\theta P\) at locations \([i, j]\) and \([i - 1, j]\). As before, an edit operation (\(D\), \(R\), or \(M\)) can be appended to the transcript to accommodate for the extra symbol \(\theta\) of the pattern sequence.

Lemma 3.4. If \((P, i, j, \epsilon)\) is an occurrence of \(P\) in data sequence \(S\) and \(\tau\) is one of its transcripts, then:

1. \((\theta P, i, j, \epsilon + 1)\) is an occurrence of \(\theta P\) in \(S\) and \(\tau_D\) is one of its transcripts,
2. \((\theta P, i - 1, j, \epsilon)\) is an occurrence of \(\theta P\) in \(S\) and \(\tau_M\) is one of its transcripts, if \(S_{[i-1]} = \theta\),
3. \((\theta P, i - 1, j, \epsilon + 1)\) is an occurrence of \(\theta P\) in \(S\) and \(\tau_R\) is one of its transcripts, if \(S_{[i-1]} \neq \theta\).

Proof. Similar to that of Lemma 3.3.
3.1.3.2 Prefix and Suffix Expansions

Based on the lemmas of the previous chapter, we next show that it is possible to construct the set of minimal occurrences of a region from those of a smaller one. Assume a data sequence \( S \), a pattern \( P \), and a cost threshold \( k \). Let \( O \) be the set of minimal occurrences of \( P \) in \( S \) with transcript cost not more than \( k \). In the following, we describe a set of expansion rules that, when applied to \( O \), produce the minimal occurrences of patterns \( P\gamma \) and \( \theta P \), where \( \gamma, \theta \in \Sigma \). We describe two sets of expansion rules: prefix and suffix expansion rules.

First, we present the prefix expansion rules. Consider the case of pattern \( P\gamma \), which has \( P \) as prefix.

**Definition 3.2 (Prefix Expansion).** The prefix expansion of \( O \) with symbol \( \gamma \in \Sigma \), denoted as \( O\gamma \), is a set of occurrences of pattern \( P\gamma \) in \( S \), with cost not more than \( k \), derived according to the following expansion rules.

For each \((P, i, j, \epsilon) \in O\):

1. If \( \epsilon + 1 \leq k \), insert into \( O\gamma \) the occurrences \((P\gamma, i, j + x, \epsilon + x + 1)\) for all \( 0 \leq x \leq k - \epsilon - 1 \).

2. If \( \epsilon \leq k \) and \( S_{j+1} = \gamma \), insert into \( O\gamma \) the occurrences \((P\gamma, i, j + x + 1, \epsilon + x)\) for all \( 0 \leq x \leq k - \epsilon \).

3. If \( \epsilon + 1 \leq k \) and \( S_{j+1} \neq \gamma \), insert into \( O\gamma \) the occurrences \((P\gamma, i, j + x + 1, \epsilon + x + 1)\) for all \( 0 \leq x \leq k - \epsilon - 1 \).

During occurrence insertion, if another in \( O\gamma \) occurs at the same location, keep the one with the smallest transcript cost.

Intuitively, these rules apply cases 1, 2, or 3 of Lemma 3.3, respectively, to occurrence \((P, i, j, \epsilon)\), and, then, apply Lemma 3.1 repeatedly (once per \( x \) value so as not to exceed the error threshold \( k \)) to each derived occurrence of \( P\gamma \).

As an example, consider the data sequence \( S \) of Figure 3.3 and let \( P = \text{GCCA}, \gamma = \text{T} \). \((P, 13, 16, 0)\) is one minimal occurrence of \( P \) in \( S \) with cost not more than \( k = 1 \). Expansion rule 1 on occurrence \((P, 13, 16, 0)\) produces \((P\gamma, 13, 16, 1)\), while rule 2 produces \((P\gamma, 13, 17, 0)\) and \((P\gamma, 13, 18, 1)\). Note that expansion rule 3 does not apply, since \( S_{17} = \gamma = \text{T} \).

The next theorem shows that the prefix expansion rules are sound, i.e., they produce occurrences of \( P\gamma \) that are minimal, and complete, i.e., they produce all minimal occurrences of \( P\gamma \).

**Theorem 3.1.** If \( O \) is the set of all minimal occurrences of \( P \) in \( S \) with transcript cost not more than \( k \), then its prefix expansion \( O\gamma \) is the set of all minimal occurrences of \( P\gamma \) in \( S \) with transcript cost not more than \( k \).

**Proof.** Let \( O' \) be the set of all minimal occurrences of \( P\gamma \) in \( S \) with transcript cost not more than \( k \). We first show that \( O' \subseteq O\gamma \) by contradiction.

Suppose there is a minimal occurrence \((P\gamma, i, j, \epsilon)\) of \( P\gamma \) with \( \epsilon \leq k \) which does not appear in \( O\gamma \). Let \( \tau \) be any of the edit transcripts of this occurrence. There are four cases based on the last operation in \( \tau \). Let \( \tau' \) denote the transcript obtained by omitting this last operation.

23
Assume the last operation is $D$, i.e., $\tau = \tau'D$. It is easy to see that $c(\tau') = \epsilon - 1$ and that $(P, i, j, \epsilon - 1)$ is an occurrence of $P$. We show that this occurrence is minimal.

Suppose otherwise. Then, there exists a transcript $\tau*$ that corresponds to an occurrence $(P, i, j, c(\tau*))$, where $c(\tau*) < c(\tau') = \epsilon - 1$. According to case 1 of Lemma 3.3, $(P\gamma, i, j, c(\tau*) + 1)$ is an occurrence of $P\gamma$ with transcript $\tau* D$ and cost $c(\tau*) + 1 < \epsilon$. However, this is not possible, as the minimal occurrence of $P\gamma$ at location $[i, j]$ has cost $\epsilon$, as assumed. Hence, $(P, i, j, \epsilon - 1)$ is a minimal occurrence of $P$.

Consequently, according to the first rule (for $x = 0$), $(P, i, j, \epsilon - 1)$ is expanded to occurrence $(P\gamma, i, j, \epsilon)$. However, the latter was assumed to not appear in $O^\gamma$, which is a contradiction.

Using similar reasoning, one can show that this contradiction appears for the cases when the last operation is $R$ and $M$, using cases 3 and 2 of Lemma 3.3, respectively.

We have to show a contradiction for the last case, when the last operation in transcript $\tau$ is $I$. Let $y$ be the largest integer such that the last $y$ operations in $\tau$ are all insertions. Further, let $\tau_1$ be the transcript obtained from $\tau$ by omitting those last $y$ operations. Observe that $(P\gamma, i, j - y, c(\tau_1))$ has to be a minimal occurrence of $P\gamma$ at location $[i, j - y]$ with transcript cost $c(\tau_1) = \epsilon - y$; otherwise, one can construct an occurrence of $P\gamma$ at location $[i, j]$ with cost lower than the minimum $\epsilon$.

Depending on the last operation in $\tau_1$ (which cannot be $I$), one can construct a minimal occurrence of $P$ from $(P\gamma, i, j - y, c(\tau_1))$ in a manner similar to the three cases examined before. Then, applying the corresponding expansion rule setting $x = y$, we obtain that $(P\gamma, i, j, \epsilon)$ appears in $O^\gamma$, i.e., a contradiction. Therefore, $O' \subseteq O^\gamma$.

Finally, we show that $O' \supseteq O^\gamma$. Suppose otherwise, i.e., there exists an occurrence $(P\gamma, i, j, \epsilon)$ of $O^\gamma$ that it is not in $O'$. Since this occurrence is not minimal, there must exist another, say $(P\gamma, i, j, \epsilon') \in O'$, with $\epsilon' < \epsilon$. We have already shown that $O' \subseteq O^\gamma$, which implies that $(P\gamma, i, j, \epsilon')$ is also in $O^\gamma$. As a result, both $(P\gamma, i, j, \epsilon)$ and $(P\gamma, i, j, \epsilon')$ are in $O^\gamma$. This is a contradiction because the expansion rules dictate that only the occurrence with the smallest cost among those occurring at the same location is allowed in $O^\gamma$.

Finally, consider the case of pattern $\theta P$, which has $P$ as suffix.

**Definition 3.3 (Suffix Expansion).** The suffix expansion of $O$ with symbol $\theta \in \Sigma$, denoted as $^\theta O$, contains a set of occurrences of pattern $\theta P$ in $S$ with cost not more than $k$, and is derived according to the following expansion rules.

For each $(P, i, j, \epsilon) \in O$:

1. If $\epsilon + 1 \leq k$, insert into $^\theta O$ the occurrences $(\theta P, i - x, j, \epsilon + x + 1)$ for all $0 \leq x \leq k - \epsilon - 1$.

2. If $\epsilon \leq k$ and $S_{i-1} = \theta$, insert into $^\theta O$ the occurrences $(\theta P, i - x - 1, j, \epsilon + x)$ for all $0 \leq x \leq k - \epsilon$.

3. If $\epsilon + 1 \leq k$ and $S_{i-1} \neq \theta$, insert into $^\theta O$ the occurrences $(\theta P, i - x - 1, j, \epsilon + x + 1)$ for all $0 \leq x \leq k - \epsilon - 1$.
During occurrence insertion, if another in \( \theta \mathcal{O} \) occurs at the same location, keep the one with the smallest transcript cost.

Intuitively, these rules apply cases 1, 2, or 3 of Lemma 3.4, respectively, to occurrence \((P, i, j, \epsilon)\), and, then, apply Lemma 3.2 repeatedly (once per \( x \) value so as not to exceed the error threshold \( k \)) to each derived occurrence of \( \theta P \). The next theorem shows the soundness and completeness of the suffix expansion rules.

**Theorem 3.2.** If \( \mathcal{O} \) is the set of all minimal occurrences of \( P \) in \( S \) with transcript cost not more than \( k \), then its suffix expansion \( \theta \mathcal{O} \) is the set of all minimal occurrences of \( \theta P \) in \( S \) with transcript cost not more than \( k \).

**Proof.** Similar to that of Theorem 3.1.

Note that consecutive applications of the prefix and suffix expansion rules can produce the minimal occurrences of a pattern from the minimal occurrences of one of its subsequences, as shown in the next theorem.

**Theorem 3.3.** Given two patterns \( P, P' \) such that \( P \subset P' \), and the set \( \mathcal{O} \) of all minimal occurrences of \( P \) in \( S \) with transcript cost not more than \( k \), it is possible to construct the set of all minimal occurrences of \( P' \) in \( S \) with transcript cost not more than \( k' \), for any \( k' \leq k \).

**Proof.** Observe that for any pattern the set of its occurrences with cost not more than \( k \) is a superset of the set of its occurrences with cost not more than \( k' \), where \( k' \leq k \). Therefore, we only need to prove that the set of minimal occurrences of \( P' \) with transcript cost not more than \( k \) can be obtained from \( \mathcal{O} \).

Since \( P \subset P' \), there exists a sequence of patterns \( P_1, \ldots, P_n \), such that \( P_1 = P \), \( P_n = P' \), and either \( P_{i+1} = P_i \gamma \) or \( P_{i+1} = \theta P_i \) holds, where \( \gamma, \theta \in \Sigma \). Due to Theorems 3.1 and 3.2, an application of the appropriate (prefix if \( P_{i+1} = P_i \gamma \), suffix otherwise) expansion rules to the set of minimal occurrences of \( P_i \) constructs the set of all minimal occurrences of \( P_{i+1} \). After successive applications, the required set can be constructed.

### 3.1.4 The PS-ARSM Method

The Prefix-Suffix ARSM (PS-ARSM) method exploits the overlaps among regions and applies the expansion rules of Chapter 3.1.3.2 to efficiently produce all ARSM results. The key idea is to initially determine the minimal occurrences for the smallest possible region, the core, and then progressively expand them to construct the minimal occurrences for all regions. Special care is required so that the produced occurrences obey the two requirements set in Definition 3.1. Note that since all occurrences produced by our method are minimal, we drop the characterization minimal in the remainder of this paper.

We first introduce some important concepts in Chapter 3.1.4.1. Then, we describe the PS-ARSM algorithm in Chapter 3.1.4.2, and detail its implementation in Chapter 3.1.4.3. Finally, we present a cost analysis of PS-ARSM and propose an optimization in Chapter 3.1.5. For ease of reference, we include the most common symbols and their definitions in Table 3.1.
Table 3.1: Common notation.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ</td>
<td>Alphabet</td>
</tr>
<tr>
<td>S</td>
<td>Data sequence</td>
</tr>
<tr>
<td>P</td>
<td>Pattern</td>
</tr>
<tr>
<td>R</td>
<td>A region of P</td>
</tr>
<tr>
<td>C</td>
<td>Core of P</td>
</tr>
<tr>
<td>a</td>
<td>Start position of C in P</td>
</tr>
<tr>
<td>b</td>
<td>End position of C in P</td>
</tr>
<tr>
<td>K()</td>
<td>Threshold function</td>
</tr>
<tr>
<td>c</td>
<td>A suffix chain in the region lattice</td>
</tr>
<tr>
<td>sc_num</td>
<td>Number of suffix chains</td>
</tr>
<tr>
<td>sc_length</td>
<td>Length of the suffix chains</td>
</tr>
</tbody>
</table>

3.1.4.1 Region Lattice

PS-ARSM operates on the region lattice induced by the subsequence relation ⊏. Figure 3.4 presents the region lattice for the example of Figure 3.3. The top left region $R_1$ corresponds to the core, while the bottom right $R_{15}$ to the pattern. A horizontal (resp. vertical) arrow from a region to another one implies that the former is a suffix (resp. prefix) of the latter.

A head is a region such that none of its suffixes, except itself, are regions. A tail is a region such that it is not the suffix of any other region. In Figure 3.4, there exist three heads, $R_1$, $R_2$ and $R_4$, and three tails, $R_{12}$, $R_{14}$ and $R_{15}$. The heads and tails of a lattice are totally ordered based on the ⊏ relation. The first head is the core, while the last tail is the pattern.

A suffix chain is a totally ordered set of regions that contains a tail and all its suffixes. The suffix chains of a lattice are ordered according to the rank of their tail. There exist three suffix chains in Figure 3.4, labelled $c_1$, $c_2$, $c_3$, each corresponding to a row of the lattice. Observe that the smallest region in a suffix chain is a head and the largest is a tail; e.g., in chain $c_1 = \{R_1, R_3, R_6, R_9, R_{12}\}$, $R_1$ and $R_{12}$ are its head and tail, respectively.

3.1.4.2 Algorithm Description

PS-ARSM consists of three execution phases. We next describe these phases in detail.

Phase 1. In this phase, PS-ARSM determines the occurrences of the core with transcript cost not more that $K(\mid P\mid)$. Note that the cost threshold $K(\mid P\mid)$ is selected...
Figure 3.5: Core occurrences, head occurrences and seeds for suffix chains $c_1$, $c_2$, and $c_3$.

so that no ARSM result, produced by expansions of the core occurrences, is missed, as explained in the following.

Recall that an occurrence of a region $R$ can only be an ARSM result if its cost is not more than $K(\vert R \vert)$ (first requirement in Definition 3.1). Since the pattern $P$ is the largest region and $K$ is monotonically increasing, $K(\vert P \vert)$ is the highest cost any ARSM result is allowed to have. From Theorem 3.3, it follows that any ARSM result must be among the expansions of the core occurrences with the loosest possible cost threshold, i.e., $K(\vert P \vert)$.

For the ARSM example shown in Figure 3.3 and the corresponding lattice in Figure 3.4, the first phase of PS-ARSM computes the core occurrences, i.e., those of region $R_1 = TCCA$, with cost at most $K(\vert P \vert) = K(10) = 2$. All these occurrences are depicted at the top of Figure 3.5 as oval boxes aligned with respect to the data sequence $S$. For instance, $(R_1, 2, 4, 2)$ corresponds to an occurrence of the core $R_1$ at location $[2, 4]$ in $S$.

**Phase 2.** In this phase, PS-ARSM first applies the prefix expansion rules on the core occurrences to produce the occurrences of all heads. E.g., in the lattice of Figure 3.4, PS-ARSM prefix-expands the core occurrences to construct the occurrences of region $R_2 = TCCAT$, shown at the middle of Figure 3.5. The resulting head occurrences are then expanded to obtain those of $R_4 = TCCATC$, shown at the bottom of Figure 3.5.

Next, PS-ARSM filters the head occurrences of each suffix chain to provide the appropriate input to phase 3. Consider a chain $c$, and let $R_1^c$ and $R_n^c$ be its head and tail, respectively. Briefly, the goal of phase 3 is to produce the occurrences of any region in $c$ by expanding its head occurrences. Note that all produced occurrences have cost at most $K(\vert P \vert)$, as they are expansions of the core occurrences. However, observe that $K(\vert R_n^c \vert)$, which is not more than $K(\vert P \vert)$, is the highest cost any occurrence of a region in $c$ is allowed to have (see Definition 3.1). Therefore, from Theorem 3.3, it follows that only head occurrences with cost not more than $K(\vert R_n^c \vert)$ should be suffix-expanded in phase 3. We refer to these occurrences of $R_1^c$
as the seeds of the suffix chain $c$. For example, the head occurrences in Figure 3.5 that are also seeds are marked with an asterisk; e.g., $(R2, 2, 5, 2)$ is a seed of suffix chain $c2$. Thus, phase 2 filters out non-seeds from head occurrences to provide the phase 3 input.

**Phase 3.** In this phase, PS-ARSM produces the ARSM results. The algorithm operates holistically on all chains, but for clarity we only describe the procedure for a single chain $c$. For each region along the chain, starting from the head $R^c_1$ and ending at the tail $R^c_n$, PS-ARSM performs the following tasks.

Suppose $R^c_i$ is the current region. PS-ARSM first suffix-expands the occurrences of the previous region $R^c_{i-1}$ in the chain to produce the occurrences of $R^c_i$. These expanded occurrences are called candidates, and the candidates of $R^c_1$ are the seeds. Then, PS-ARSM enforces the two requirements of Definition 3.1. For the first, it excludes candidates with cost more than $K(|R^c_i|)$; the remaining are to be inserted in the result set. However, during insertion, PS-ARSM removes any occurrence (either a candidate, or one already in the result set) that violates the second requirement. It is important to note that all candidates for $R^c_i$ (i.e., not only those inserted in the result set) are required to obtain the candidates for the next region $R^c_{i+1}$ in the chain.

Figure 3.6 illustrates the third phase for all the suffix chains of the lattice shown in Figure 3.4. Consider suffix chain $c_2$. Its head is $R^{c_2}_1 = R2$ and the tail is $R^{c_2}_n = R14$. The seeds of this chain, i.e., the occurrences of $R2$ with cost at most $K(|R14|) = K(9) = 2$, have been determined in the second phase. Assume that the currently examined region is $R5$. Its four candidates (see the second part of Figure 3.6) are produced by the suffix expansion of $R2$ candidates (at the middle of the first part of Figure 3.6). Then PS-ARSM considers only those candidates with cost 1 (since $K(|R5|) = K(6) = 1$) for insertion in the result set. In our example, this is an empty set.

**Pseudocode.** Figure 3.7 presents the PS-ARSM pseudocode. The algorithm takes as input the data sequence $S$, the pattern $P$, the core $[a, b]$ and the threshold function $K$, and outputs the ARSM results. In the first phase, PS-ARSM applies an ASM algorithm to compute the core occurrences (line 1); details are discussed in Chapter 3.1.4.3.

In the second phase (lines 2–6), PS-ARSM constructs the head occurrences. The occurrences of the first head are those of the core (line 2). The occurrences of each other head are obtained from those of the previous head (line 4). The pExp method implements the prefix expansion rules. Then, PS-ARSM identifies the seeds for the corresponding suffix chain, by invoking the getSeeds method (line 5). Both methods are discussed in Chapter 3.1.4.3.

In the third phase (lines 7–14), PS-ARSM computes the ARSM results. For clarity, pseudocode presents the required steps independently for each suffix chain $c_j$. However, these steps are performed holistically for the seeds of all suffix chains (see Chapter 3.1.4.3 for details). Back to pseudocode, PS-ARSM first constructs the candidates, i.e., the occurrences of each region in the chain. Note that the candidates of the head are the seeds of the chain (line 8). The candidates for each other region are obtained from those of the previous region (line 10). In particular, the sExp method (presented in Chapter 3.1.4.3) implements the suffix expansion rules. Then, among the candidates of region $R^c_j$, those with cost not more than $K(|R^c_j|)$ are identified by the sieve method (line 11 and Chapter 3.1.4.3). Finally, the remaining
occurrences are inserted in the result set, enforcing the second requirement of Definition 3.1 (line 12).

Correctness. PS-ARSM applies Theorem 3.3, i.e., it implements prefix and suffix rules, which are sound and complete as we proved. The previous guarantees that PS-ARSM will return all the ARSM results.

3.1.4.3 Implementation

An important observation of PS-ARSM is that prefix (resp. suffix) expanding a set of occurrences that end (resp. start) at the same position in the data sequence, requires the same computations. To understand this, refer to Figure 3.5, and consider the core occurrences \((R_1, 1, 5, 2)\) and \((R_1, 2, 5, 1)\), which both end at position 5. We describe the operations necessary to prefix expand these occurrences and obtain the head occurrences of region \(R_2\). First, observe that \(R2 = R1T\), and that the next symbol in the data sequence, i.e., at position 6, is A. Then, applying Lemma 3.3 for \(P = R1, i \in \{1, 2\}, j = 5, \epsilon \in \{2, 1\}\), and since \(S[j+1] = A \neq T = \gamma\), we obtain that each core occurrence results in two head occurrences: \((R2, 1, 5, 3)\), \((R2, 1, 6, 3)\) from the first, and \((R2, 2, 5, 2)\), \((R2, 2, 6, 2)\) from the second. Observe that these
PS-ARSM()
Input: $S, P, [a,b], K$
Output: results
begin
# Phase 1
01. core\_occs $\leftarrow$ ASMfull$(S, P_{[a,b]}, K(|P|))$
# Phase 2
02. head\_occs$_{c_j}$ $\leftarrow$ core\_occs
03. foreach suffix chain $c_j$
04. if $(j > 1)$ then head\_occs$_{c_j}$ $\leftarrow$ pExp(head\_occs$_{c_{j-1}}$, $R_{i}^j$)
05. seeds$_{c_j}$ $\leftarrow$ getSeeds(head\_occs$_{c_j}$, $K(|c_j.tail|)$)
06. end
# Phase 3
07. foreach suffix chain $c_j$
08. cand$_s_1$ $\leftarrow$ seeds$_{c_j}$
09. foreach region $R_{i}^j$ of $c_j$
10. if $(i > 1)$ then cand$_s_i$ $\leftarrow$ sExp(cand$_s_{i-1}$, $R_{i}^j$)
11. tmp $\leftarrow$ sieve(cand$_s_i$, $K(|R_{i}^j|)$)
12. insert(results, tmp)
13. end
14. end
end.

Figure 3.7: The PS-ARSM algorithm.

can be succinctly represented as the two head occurrences $(R2, i, 5 + 0, \epsilon + 1)$ and $(R2, i, 5 + 1, \epsilon + 1)$, for $i \in \{1, 2\}$, $\epsilon \in \{2, 1\}$. It is important to notice here that the information we have underlined in the previous representation (i.e., +0, +1 for the end position, and +1, +1 for the cost) completely describes how the head occurrences are constructed from the core occurrences.

Based on this observation, PS-ARSM employs data structures that serve two goals:

1. they compactly represent occurrences, and
2. they facilitate prefix and suffix expansions by avoiding redundant computations.

**Hash table head\_occs.** The hash table head\_occs is used to compute and store the head occurrences during the second phase of PS-ARSM. Initially, it contains the core occurrences, i.e., the head occurrences of the first suffix chain. Then, its entries are updated sc\_num $- 1$ times, where sc\_num is the number of suffix chains in the region lattice. After the $(i - 1)$-th update, head\_occs contains the head occurrences for the $i$-th suffix chain.

In what follows, we describe the head\_occs contents for suffix chain $c_i$; the update procedure is explained later. head\_occs consists of key-value entries. A key corresponds to the end position of a core occurrence in the data sequence. The value for key $j$, denoted as head\_occs[$j$], is a composite value with information about the head occurrences of $c_i$ produced by the prefix-expansion of core occurrences ending at position $j$. In particular, this composite value consists of:

- **core\_occs**: the list of core occurrences ending at position $j$, ordered by their transcript cost.

- **add\_cost**: an array where the $x$-th entry denotes the additional transcript cost required for the expansion of any core occurrence in core\_occs to a head occurrence of $c_i$ ending at position $j + x$.

Note that the length of the add\_cost array is sc\_num $+ K(|P|)$, as this value is the
farthest a core occurrence can be expanded to the right while its transcript cost remains not more than \( \mathcal{K}(|P|) \).

Observe that head\_occs indirectly describes the head occurrences of suffix chain \( c_i \). That is, it stores the core occurrences (core\_occs), and how to expand them (add\_cost) so as to produce the head occurrences. We illustrate the above using an example. Assume the lattice of Figure 3.4 and consider head\_occs after the first update, i.e., containing the head occurrences of suffix chain \( c_2 \). Figure 3.8 depicts the contents of entry head\_occs[5]. core\_occs contains all the core occurrences ending at position 5 in the data sequence \( S \). Note that an add\_cost array is represented as two rows: the lower shows the contents, while the upper presents the indices of this zero-based array. The depicted add\_cost informs us that head occurrences of \( c_2 \) (produced by any core occurrence in core\_occs) ending at positions 5 + 0 = 5 and 5 + 1 = 6, have additional cost of 1. Moreover, any head occurrence ending at subsequent positions, e.g., 5 + 2 = 7, has an additional cost larger than \( \mathcal{K}(|P|) \), denoted as * in Figure 3.8. Therefore, the head occurrences in head\_occs[5] are \((R2,2,5,2)\), \((R2,2,6,2)\), \((R2,3,5,2)\) and \((R2,3,6,2)\), which are all head occurrences of \( c_2 \) that are expansions of core occurrences ending at position 5.

**Hash table \texttt{cands}.** The hash table \texttt{cands} is used to compute and store the candidates during the third phase of PS-ARSM. Initially, it contains the seeds of all the suffix chains, i.e., the candidates for all the heads. Then, its entries are updated \( sc_{\text{length}} - 1 \) times, where \( sc_{\text{length}} \) is the length of any suffix chain in the region lattice. After the \((i+1)\)-th update, \texttt{cands} contains the candidates for the \( i \)-th regions of all suffix chains.

In what follows, we describe the contents of \texttt{cands} for the \( i \)-th regions of all suffix chains; the update procedure is explained later. \texttt{cands} consists of key-value pairs. A **key** corresponds to the start position of a seed in the data sequence. The **value** for key \( j \), denoted as \texttt{cands}[j], is a composite value with information about the candidates produced by the suffix-expansion of seeds starting at position \( j \). In particular, this composite value consists of:

- **seeds**: the list of seeds starting at position \( j \), ordered by their transcript cost.
- **add\_cost**: an array where the \( x \)-th entry denotes the additional transcript cost required for the expansion of any seed in \texttt{seeds} to a candidate starting at position \( j - x \).

Note that the length of the add\_cost array is \( sc_{\text{length}} + \mathcal{K}(|P|) \), as this value is the farthest a seed can be expanded to the left while its transcript cost remains not more than \( \mathcal{K}(|P|) \).
### Figure 3.9: An excerpt of the cands hash table, containing the candidates for $R_3$, $R_5$, and $R_7$.

Similar to `head_occs`, `cands` indirectly describes the candidates. That is, it stores the seeds (`seeds`), and how to expand them (`add_cost`) so as to produce the candidates. We illustrate the above using the example lattice of Figure 3.4. Assume `cands` after the first update, containing the candidates for regions $R_3$, $R_5$, and $R_7$. Figure 3.9 depicts the entry `cands[12]`. Observe that seed $(R_2, 12, 17, 2)$ produces the single candidate $(R_5, 11, 17, 2)$, as explained in the figure, since all other occurrences exceed the cost threshold.

The goal of phase 1 is to produce all core occurrences with cost at most $\mathcal{K}(|P|)$. Note that an off-the-shelf ASM algorithm (like those discussed in Chapter 2.1.1.3) cannot produce all core occurrences. This happens because, by design, all ASM algorithms ignore some of the overlapping occurrences. For instance, if two occurrences have the same end position but different transcript costs, only the one with the lowest cost is reported. To illustrate this assume the classic dynamic programming algorithm described in [26]. Consider the core occurrence $(R_1, 12, 16, 2)$ in Figure 3.5. It would not be among the results, because $(R_1, 13, 16, 1)$ and $(R_1, 14, 16, 1)$ end at the same position but have better transcript cost. However, $(R_1, 12, 16, 2)$ should not be discarded, as it produces (via expansions in the last two phases) $(R_{14}, 8, 17, 2)$, which is an ARSM result.

In order to produce all core occurrences, we follow three steps: (1) we execute a conventional ASM algorithm to discover the endpoints of the occurrences, (2) we construct disjoint windows around these endpoints, so that any occurrence is completely located within a window, and (3) in each window, we execute a variation of the dynamic programming algorithm (described below) to efficiently produce all occurrences.

Note that any ASM algorithm, including index-based solutions, can be used in step 1. Subsequently, in step 2, we construct the window $[i - |C| - \mathcal{K}(|P|) + 1, i]$ for each endpoint $i$ found at the previous step. This is because it is impossible for an occurrence to start before the $(i - |C| - \mathcal{K}(|P|) + 1)$-th position. To avoid redundant computations, we merge any overlapping windows.

Finally, in step 3, we execute a variation of the dynamic programming algorithm in [96] that is specifically adapted to return all occurrences. Recall that in conventional algorithms, each dynamic programming cell $DP[i, j]$ contains the cost and the start position of the best occurrence of the pattern prefix $P_{[1,i]}$ ending at position $j$ in the data sequence $S$. On the other hand, in our variant, we keep in each cell $DP[i, j]$ an array which contains the costs of the best occurrences of $P_{[1,i]}$ ending at position $j$ for each possible start position in $S$. Finally, since conventional algorithms cannot retrieve occurrences whose transcripts start with Insert operations, we must take special care so as not to miss them.

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>$(R_2, 12, 17, 2)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>seeds</th>
<th>add_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[1, 0, 1, * * * *]$</td>
<td>$[2, 3, 4, 5, 6, 1, 0, 1, * * * *]$</td>
</tr>
</tbody>
</table>

(R5, 12-1, 17, 2+0)
The second phase of PS-ARSM produces the seeds of all suffix chains, by progressively prefix-expanding head occurrences. First, head_occ is initialized with the core occurrences found in Phase 1. Then, PS-ARSM proceeds in \(sc_{num} - 1\) iterations. In each iteration, PS-ARSM executes two tasks: (1) it invokes \(pExp\) to update head_occ so as to contain the head occurrences of the next suffix chain, and (2) it invokes getSeeds to identify the seeds and initialize the cands hash table, which is required for Phase 3.

We now describe the \(pExp\) method. Assume that head_occ contains the head occurrences of suffix chain \(c_i\). Then, \(pExp\) updates head_occ (by modifying the add_cost arrays) so that it contains the head occurrences of \(c_{i+1}\). In particular, \(pExp\) visits a hash entry of head_occ and applies the prefix expansion rules for each possible end position of an occurrence. Fix a head_occ entry \(h\) with key \(h.key\), and let \(\gamma\) be the last symbol in the head (first region) of suffix chain \(c_{i+1}\). Let add_cost\((c_i)\) (resp. add_cost\((c_{i+1})\)) denote the array for suffix chain \(c_i\) (resp. \(c_{i+1}\)). Initially add_cost\((c_{i+1})\) is filled with * values. Then, \(pExp\) scans add_cost\((c_i)\) and applies the following procedure for each entry until a * value is encountered. Consider the \(j\)-th entry in add_cost\((c_i)\). Recall that this represents head occurrences of \(c_i\) that end at position \(h.key + j\). First, the update procedure applies Lemma 3.3 (cases 1, 2 if symbols \(\gamma\) and \(S[h.key+j]\) match, or cases 1, 3 otherwise) to compute the additional costs of head occurrences of \(c_{i+1}\) that end at positions \(h.key + j\) and \(h.key + j + 1\). If the corresponding entries in add_cost\((c_{i+1})\) have higher additional costs, they are updated. Finally, \(pExp\) applies Lemma 3.1 for each entry of add_cost\((c_{i+1})\). As before, it only updates an entry when the computed additional cost is smaller than the entry’s existing value.

Figure 3.10 illustrates an application of \(pExp\), assuming the maximum allowed cost is 2. The left column shows the add_cost array for suffix chain \(c_i\), while the right column shows how add_cost array for the next suffix chain \(c_{i+1}\) is updated. The first row shows that add_cost\((c_{i+1})\) has initially all * values. Then, \(pExp\) examines entry add_cost\((c_i)[0]\), and applies cases 1 and 3 of Lemma 3.3 \((S[h.key+1] = A \neq T = \gamma)\).
This implies that entries 0 and 1 of $\text{add}\_\text{cost}(c_{i+1})$ have cost 1 more than that in $\text{add}\_\text{cost}(c_{i})[0]$. Note that Lemma 3.1 does not update any $\text{add}\_\text{cost}(c_{i+1})$ entry as it would get additional cost larger that the maximum allowed.

Next, $\text{pExp}$ examines entry $\text{add}\_\text{cost}(c_{i})[1]$ (see third row of Figure 3.10). This time, cases 1 and 2 of Lemma 3.3 apply. Therefore, entry 1 of $\text{add}\_\text{cost}(c_{i+1})$ is updated with additional cost $0 + 1 = 1$ (case 1), as it is lower than its current value. On the other hand, entry 2 of $\text{add}\_\text{cost}(c_{i+1})$ has additional cost $0 + 0 = 0$ (case 2). Lemma 3.1 fills each remaining entry with additional costs 1 more compared to the previous entry.

Cases 1 and 3 of Lemma 3.3 apply for $\text{add}\_\text{cost}(c_{i})[2]$. However, since they produce occurrences with costs worse than those produced in the previous step, $\text{add}\_\text{cost}(c_{i+1})$ is not updated (see fourth row of Figure 3.10). The $\text{pExp}$ method continues with the next entries and terminates when it reaches $*$ in the final entry.

Finally, we describe the $\text{getSeeds}$ method. Its goal is to identify the seeds among the head occurrences of the current suffix chain. Using the information in the $\text{core}\_\text{oocs}$ and $\text{add}\_\text{cost}$ fields, $\text{getSeeds}$ selects those head occurrences with cost not more than the allowable for the current suffix chain. The selected occurrences are the seeds, and are inserted in the $\text{cands}$ hash table.

The third phase of $\text{PS-ARSM}$ produces the actual ARSM results, by progressively suffix-expanding the seeds of all suffix chains. Note that Phase 2 has initialized hash table $\text{cands}$ with all seeds. Then, $\text{PS-ARSM}$ proceeds in $sc\_\text{length}$ iterations. In each iteration, $\text{PS-ARSM}$ executes two tasks: (1) it invokes $\text{sExp}$ (except in the first iteration) to update $\text{cands}$ so as to contain the candidates of the next region of all suffix chains, and (2) it invokes $\text{sieve}$ and $\text{insert}$ to produce the actual ARSM results.

The $\text{sExp}$ method is similar to $\text{pExp}$, except the following differences: (1) it operates on hash table $\text{cands}$, (2) it applies Lemmas 3.4 and 3.2, and (3) it examines the data sequence backwards. In the interest of space, we do not detail its operations.

The $\text{sieve}$ method identifies ARSM results among the current candidates in the hash table $\text{cands}$. Finally, $\text{insert}$ adds these occurrences to the set of ARSM results, taking care so that the second requirement of Definition 3.1 is not violated.

### 3.1.5 Cost Analysis and Optimization

**Cost of Phase 1.** The first phase involves three steps. In the first two, a conventional ASM algorithm is used to mark windows of the data sequence $S$ that contain core occurrences with cost not more than $K(|P|)$. Assume that the dynamic programming algorithm with cut-off heuristic [96] is applied. Then, according to [8], the average processing time of the first two steps is at most $\left(\frac{K(|P|)}{1-e/\sqrt{K(|P|)}} + O(1)\right) \cdot |S| \cdot T_{DP} = O(K(|P|) \cdot |S|)$, where $T_{DP}$ is the required time to compute each dynamic programming cell (constant for a given system configuration).

In the third step of Phase 1, our dynamic programming variation is executed for each window marked in the previous steps. In our analysis, we assume the worst case scenario, where the entire data sequence is marked as a single window. Following again the analysis in [8], this step requires $O(K(|P|)^2 \cdot |S|)$ time on average, because each dynamic programming cell contains $2 \cdot K(|P|) + 1$ values.

Putting everything together, Phase 1 has a total processing time of $O(K(|P|)^2 \cdot$.
Cost of Phases 2 and 3. To determine the cost of the last two phases, observe that each core occurrence, which is computed in the first phase, will undergo the same number of (prefix and suffix) expansions. In particular, the number of prefix expansions is one less than the number of suffix chains, i.e., $|P| - \beta$. On the other hand, the number of suffix expansions is one less that the length of a suffix chain, i.e., $\alpha - 1$. In total, a core occurrence will undergo $|P| - \beta + \alpha - 1 = |P| - |C|$ expansions.

Furthermore, the total number of core occurrences produced in Phase 1 is given by $|S| \cdot f(|C|, K(|P|))$, where $f(|C|, K(|P|))$ is the probability of a random sequence of length $|C|$ matching in a given position of the data sequence with transcript cost not more than $K(|P|)$.

Putting everything together, and assuming that each expansion requires $T_{EX}$ time (constant for a given system configuration), the processing time of Phases 2 and 3 is $|S| \cdot f(|C|, K(|P|)) \cdot (|P| - |C|) \cdot T_{EX} = O(|S| \cdot f(|C|, K(|P|)) \cdot (|P| - |C|))$.

Note that computing a closed formula for the function $f()$ is a difficult task [8]. However, it is possible to derive the following upper bound for $f(|C|, K(|P|))$ based on the analysis in the appendix of [8]:

$$
\begin{align*}
&\sum_{i=|C|-K(|P|)}^{|C|} \frac{1}{\sum_{i=1}^{|C|-K(|P|)} i} \left( \frac{|C|}{|C| - K(|P|)} \right) \left( \frac{i}{i - K(|P|)} \right) \\
&+ \sum_{i=|C|+1}^{|C|+K(|P|)} \frac{1}{\sum_{i=1}^{|C|+K(|P|)} i} \left( \frac{|C|}{|C| + K(|P|)} \right) \left( \frac{i}{i + K(|P|)} \right).
\end{align*}
$$

Optimization. In some ARSM instances (e.g., when the ratio $K(|P|)/|R|$ is large), it is possible that the number of core occurrences is so high that the total execution time of PS-ARSM is dominated by Phases 2 and 3. For such instances, it is often preferable to divide the problem into two ARSM sub-problems, where each has much fewer core occurrences than the original, and solve them independently. This can be achieved by splitting the lattice in two, and appropriately defining the pattern sequence and the core for each sub-problem — the threshold function $K$ is common. Figure 3.11 shows the two ways (vertical and horizontal partition) to view an ARSM problem as two independent sub-problems. The core and pattern regions for each sub-problem are also illustrated. Note that at the end, the occurrences from one sub-problem must be checked against those of the other, so as to remove occurrences that violate the second requirement of Definition 3.1.

The only question that remains is when and how to perform this sub-problem division. Based on the analysis of the previous paragraphs we estimate the total execution time of three scenarios: (a) solving the original problem, (b) solving the
two sub-problems produced by a vertical split of the lattice, and (c) solving the two sub-problems produced by a horizontal split of the lattice. The scenario having the smallest estimated cost is the one selected.

### 3.1.6 Experimental evaluation

We run a comprehensive set of experiments to assess the performance of our PS-ARSM method on both synthetic and real datasets. Chapter 3.1.6.1 describes the experimental setup and Chapter 3.1.6.2 presents our findings.

#### 3.1.6.1 Setup

**Algorithms.** The evaluation involves three exact algorithms, i.e., they correctly retrieve all ARSM results.

- **PS-ARSM**, our proposed solution for ARSM. The dynamic programming with the cut-off heuristic algorithm [96] is used in Phase 1.

- **N-ARSM**, the naïve approach that executes an ASM dynamic programming algorithm for every single region. The cut-off heuristic [96] is used to improve the dynamic programming efficiency.

- **M-ARSM**, which executes MASM [22] for each group of regions that have the same length. Experiments have shown this to be the most efficient MASM-based algorithm (see also Chapter ??).

We implement all algorithms in C++, and run the experiments on a dedicated Linux PC Intel Core 2 Duo CPU, E8400, at 3.00GHz. In our system, \( T_{DP} = 3.14 \cdot 10^{-5}\text{msec} \) and \( T_{EX} = 2.11 \cdot 10^{-5}\text{msec} \).

**Parameters.** We measure performance in terms of the total time required to produce the ARSM results, while we vary the following set of parameters: the pattern sequence length \(|P|\) (measured in number of symbols); the data sequence length \(|S|\); the ratio of the core to pattern length \(|core|/|P|\); the position of the core \(cPos\); the ratio of the allowable transcript cost for each region over its length \(\alpha = K(|R|)/|R|\) (we consider linear threshold functions); the size of the alphabet \(|\Sigma|\). Table 3.2 contains all parameters and their range of examined values. In each experiment, we vary a single parameter and set the remaining to the default values shown on the table.

**Datasets.** We use synthetic and real datasets. For the synthetic (D1), we use random sequences that follow the uniform Bernoulli model, i.e., each symbol has

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Range of values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>P</td>
<td>)</td>
</tr>
<tr>
<td>(</td>
<td>S</td>
<td>)</td>
</tr>
<tr>
<td>(</td>
<td>P</td>
<td>/</td>
</tr>
<tr>
<td>(cPos)</td>
<td>left, middle, right</td>
<td>middle</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>0.1, 0.15, 0.2, 0.25, 0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>(</td>
<td>\Sigma</td>
<td>)</td>
</tr>
</tbody>
</table>
Table 3.3: Memory consumption of PS-ARSM hash tables.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>head_occs size</th>
<th>cands size</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1 run 1</td>
<td>7.69MB (128,577 entries)</td>
<td>0.17MB (2,006 entries)</td>
</tr>
<tr>
<td>D1 run 2</td>
<td>1.61MB (27,001 entries)</td>
<td>0.03MB (255 entries)</td>
</tr>
<tr>
<td>D2 run 1</td>
<td>7.82MB (126,852 entries)</td>
<td>0.37MB (3,221 entries)</td>
</tr>
<tr>
<td>D2 run 2</td>
<td>2.11MB (33,802 entries)</td>
<td>0.18MB (944 entries)</td>
</tr>
<tr>
<td>D3 run 1</td>
<td>9.51MB (155,912 entries)</td>
<td>0.33MB (3,898 entries)</td>
</tr>
<tr>
<td>D3 run 2</td>
<td>2.52MB (41,282 entries)</td>
<td>0.07MB (698 entries)</td>
</tr>
</tbody>
</table>

We also consider real datasets, obtained from the Ensembl database\(^2\). The 3’UTR dataset (D2) is a 44 million nucleotide sequence for the 3’ untranslated region of the human gene transcripts. The CDS dataset (D3) is a 74 million nucleotide sequence for the coding region of the human gene transcripts. Note that these are genomic datasets and can only be used in experiments with alphabet size \(\Sigma = 4\). We extract 20 random subsequences from these datasets to serve as the patterns, and for each of them we execute the algorithms 5 times. As a result, every reported time value is the average of 100 executions.

### 3.1.6.2 Results

**Runtime analysis of PS-ARSM.** We investigate the runtime performance of PS-ARSM for the default experimental setting. Note that, for these parameter values, PS-ARSM chooses to split the region lattice horizontally.

Table 3.3 presents the memory occupied by the two hash tables of PS-ARSM. The two runs shown correspond to the two executions of PS-ARSM, one for each half of the original lattice. Note that the size of cands is significantly smaller than that of head_occs. The reason is that much fewer occurrences survive after Phase 2. For example, as the table suggests for the first run on D1, there exist 128,577 distinct positions where occurrences end in Phase 2, but only 2,006 positions where occurrences start in Phase 3.

Table 3.4 presents the relative time spent in each phase of PS-ARSM; the values are based on the total execution time for both runs. Phase 1 is by far the most expensive as it consumes around 97% of the total running time. Among the other two phases, Phase 3 requires more time as it expands much fewer occurrences than Phase 2 (see Table 3.3).

Table 3.4: Running time breakdown for PS-ARSM phases.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Phase 1 (%)</th>
<th>Phase 2 (%)</th>
<th>Phase 3 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>97.26</td>
<td>2.57</td>
<td>0.17</td>
</tr>
<tr>
<td>D2</td>
<td>96.70</td>
<td>2.88</td>
<td>0.42</td>
</tr>
<tr>
<td>D3</td>
<td>96.58</td>
<td>3.20</td>
<td>0.22</td>
</tr>
</tbody>
</table>

**Varying the pattern sequence length.** Figure 3.12 presents the execution times (in logarithmic scale) of all algorithms as the length of the pattern sequence \(|P|\)

\(^2\)http://www.ensembl.org/biomart/martview/
varies. The findings are similar for all datasets. As the $|P|$ grows, the number of regions and their average length increases. This explains why the execution time of M-ARSM and N-ARSM grows.

On the other hand, the execution time of PS-ARSM remains less than 10 seconds, unaffected by $|P|$. Note that as $|P|$ grows while the remaining parameters remain fixed, both the length of the core $|C|$ and the cost threshold $K(|P|)$ also grow. In particular, the difference $|C| - K(|P|) = 0.5 \cdot |P| - 0.2 \cdot |P| = 0.3 \cdot |P|$ (for the default values of Table 3.2) grows linearly with the pattern length. This difference plays a critical role in the number of core occurrences produced during the first phase of PS-ARSM. As the analysis of Chapter 3.1.5 shows, the number of core occurrences decreases rapidly as $|C| - K(|P|)$ increases. Therefore, although the execution time of Phase 1 increases, that of Phases 2 and 3 decreases with $|P|$. As a result, for large $|P|$ values, PS-ARSM is almost two orders of magnitude faster.

Figure 3.12: Varying the pattern sequence length $|P|$.

Varying the data sequence length. Figure 3.13 shows the execution time on data sequences of different lengths. Note that to construct a sequence of length $|S|$ from the real datasets, we extract the first $|S|$ symbols, and that the maximum possible length is 44M for D2 and 74M for D3. The execution time of all methods grows linearly, as the data sequence size increases. Therefore, the performance improvement of PS-ARSM over M-ARSM and N-ARSM, in all datasets and $|S|$ values, is over one order of magnitude.

Figure 3.13: Varying the data sequence length $|S|$.

Varying the core/pattern length ratio. Figure 3.14 illustrates the execution time for several values of the $|C|/|P|$ ratio. The results are similar for all datasets. As the $|C|/|P|$ ratio increases, the execution time of all methods decreases. This is because the number of regions decreases and the average region length increases. Note that the benefit of PS-ARSM over its competitors increases with the ratio as, in addition to the above, the number of core occurrences produced in its first phase decreases. The reason is that $|C|$, and thus $|C| - K(|P|)$, increases (see Chapter 3.1.5).
C chooses to split the lattice horizontally. Let \( C \) of the pattern \( m \) by substituting the core lies in the middle. The corresponding notation for the core at the left is obtained denoting the core and the pattern for the top (resp. bottom) half lattice when the reason is that, in addition to the previous, the number of core occurrences are fewer when the core lies near the edges of the pattern. A detailed explanation when the decrease in the execution time of \( \text{M-ARSM} \) is more pronounced. Briefly, the reason is that, in addition to the previous, the number of core occurrences are fewer when the core lies near the edges of the pattern. A detailed explanation when the core is at the left follows (the explanation for the other case is similar).

Consider the case of the core in the middle \( C = [a_m, b_m] \) and that at the left of the pattern \( C_l = [a_l, b_l] = [a_m - c, b_m - c] \), where \( c > 0 \). In both cases, \( \text{PS-ARSM} \) chooses to split the lattice horizontally. Let \( C^{top}_m \) and \( P^{top}_m \) (resp. \( C^{bot}_m \) and \( P^{bot}_m \)) denote the core and the pattern for the top (resp. bottom) half lattice when the core lies in the middle. The corresponding notation for the core at the left is obtained by substituting \( m \) with \( l \). Then, for the top half lattice, it holds that \( |C^{top}_m| = |C^{top}_l| \) and \( |P^{top}_l| = (|P_l| + b_l - 1)/2 = |P^{top}_m| - c/2 < |P^{top}_m| \). In other words, when the core is at the left, the top half pattern is smaller, and thus the allowable cost is also smaller, which leads to fewer core occurrences. On the other hand for the bottom half lattice, it holds that \( |P^{bot}_m| = |P^{bot}_l| \), and \( |C^{bot}_l| = (|P_l| + b_l + 1)/2 - a_l + 1 = |C^{bot}_m| + c/2 > |C^{bot}_m| \). Here, the bottom half pattern, and thus the allowable cost, is the same, but the bottom half core is larger, which again means fewer occurrences when the core is at the left. Overall, the total number of core occurrences in both lattices is smaller when the core is at the left.

Varying the cost ratio. Figure 3.16 shows the execution time as the cost threshold ratio varies. Higher \( \alpha \) values require more effort by all methods. However, the performance of \( \text{M-ARSM} \) and \( \text{PS-ARSM} \) deteriorates faster and approaches that of the naïve method. Intuitively, the reason is the following. These two methods try to filter out certain areas of the data sequence that do not contain ARSM results.
When the allowable transcript cost increases, fewer and smaller areas are excluded, and thus their filtering benefit diminishes and they behave similar to the brute force method of $N$-ARSM.

Varying the alphabet size. Although the focus of this work is on genomic databases, where $|\Sigma| = 4$, we also examine the behavior of all methods on databases with different alphabet size. In particular, we consider protein sequences, where $|\Sigma| = 20$, and ASCII text sequences, where $|\Sigma| = 94$. In this experiment, we only use synthetic data. Figure 3.17 shows the results. The execution time of all methods decreases as $|\Sigma|$ increases, because there exist fewer possible occurrences (see Chapter 3.1.5). Note that both $M$-ARSM and $PS$-ARSM become significantly more efficient than the naïve method, while the benefit of $PS$-ARSM over $M$-ARSM remains close to one order of magnitude.

Using real patterns. In the final experiment, we investigate the performance of $PS$-ARSM for the real-life problem of predicting micro-RNA bindings. Based on several biological observations (e.g., [19]), we select the following parameters. We use the 3'UTR dataset (D2) as the data sequence, since most known micro-RNA bindings are located in this gene section. Further, we randomly select 100 micro-RNA sequences from MirBase\(^3\) as the patterns. Since the most important symbols for the binding are located at the left of the micro-RNA sequences, we select $|C| = 10$, $cPos = left$, and set $\alpha = 0.2$ for all regions, except the core where the allowable transcript cost is restricted to 1.

Figure 3.18 presents the results. Note that we report the total running times for all 100 micro-RNA patterns. The findings are similar to the case of synthetic patterns. $PS$-ARSM is around one order of magnitude faster than $N$-ARSM, and more than five times faster than $M$-ARSM.

\(^3\)http://www.mirbase.org/, the database where all micro-RNA sequences are registered.
3.2 Cloud-based target prediction systems

Accelerating the sequence matching step of target prediction methods is not enough to achieve near-real time performance, since these methods also incorporate some other computational intensive processes. Distributing the involved computations in the nodes of a Cloud infrastructure can help to resolve this issue. Therefore, we designed and developed two Cloud-based miRNA target prediction systems TarCloud [97] and MR-microT [40]. The former was developed using the programming framework of Microsoft Azure, it was able to provide the targets of a given miRNA in, approximately, 5 minutes, however it was not very scalable for large number of simultaneous prediction requests. The latter was based on MapReduce, it provides the predictions of any given miRNA in less than 2 minutes, and it scales well as the number of simultaneous prediction increases (provided that there are enough resources). Chapter 3.2.1 describes TarCloud, while Chapter 3.2.2 introduces MR-microT.

3.2.1 TarCloud: target prediction on MS Azure

3.2.1.1 Technologies and system architecture

TarCloud [97] is a miRNA target prediction system, based on DIANA microT v.5 method [78] (see Chapter 2.2.2) and implemented using the Microsoft Azure Cloud platform. The aforementioned platform consists of various services commoditized through three product brands. These are (a) Windows Azure, an operating system providing scalable compute and storage facilities, (b) SQL Azure, a Cloud-based version of SQL Server, and (c) Windows Azure AppFabric, a collection of services supporting Cloud applications. The platform provides an API built on REST, HTTP and XML that allows a developer to interact with the Azure services.

TarCloud follows the DIANA microT v.5 approach to predict possible targets for a given miRNA sequence (for details about DIANA microT v.5 see Chapter 2.2.2). Based on this, a TarCloud task consists of a set of distinct jobs. Figure 3.19 presents the workflow showing the dependencies among these jobs. Each rectangle represents one independent TarCloud job. The arrows depict the data flow between the jobs. The grey vertical bars represent a fork or a join. A fork ignites concurrent activities, while a join merges the output of concurrent activities. A brief description for each of the depicted jobs follows.

---

Figure 3.18: Setting with real patterns.
Aligner (J1 and J2). The aligner component is responsible for (a) locating the MREs of the miRNA sequence, (b) computing the sequence alignment score of the miRNA seed to the MRE, and (c) computing the binding energy of the miRNA to the gene (by using RNAhybrid [79]). The aligner executes the job J1 for the 3'-UTR regions of all genes of the selected genome, and J2 for the CDS regions. Note that these two jobs are executed concurrently.

Conservation profiler (J31 ... J3N and J41 ... J4M). The conservation profiler component is responsible for checking how each MRE is preserved in the genomes of several species. Currently, TarCloud exploits up to 27 species to assess the MRE conservation profile, taking into consideration both conserved and non-conserved MREs for the estimation of the final score. The conservation profiling for all MREs that belong to the same region (3'-UTR/CDS) of the same chromosome is performed as a batch. Many batches of conservation profiling computations can performed concurrently. In Figure 3.19 conservation profiling jobs on 3'-UTR regions have the prefix ‘J3’, while the rest have the prefix ‘J4’.

Score aggregator (J5). The score aggregator component computes, for each gene, the aggregated score of all its MREs. The aggregation is a weighted sum that also considers the molecular folding (i.e., the 3-D structure of the molecules). The score aggregator executes the job J5.

The TarCloud architecture, illustrated in Figure 3.20, involves Virtual Machines
(VMs), called Job Workers, fully capable to perform any of the previous jobs. Each Job Worker listens for HTTP (REST) requests and executes the TarCloud jobs described by these requests. Any input and output data are stored mostly as BLOBs in the distributed Cloud storage of Microsoft Azure. We refer to this storage space as the BLOB storage. The number of Job Worker instances deployed for a particular configuration of TarCloud, is a design parameter crucial for the efficiency of the system. In order to distribute the HTTP requests to the Job Worker instances, we also deploy a VM, called Balancer. Note that it is not essential that the Balancer is located in a separate VM. One of the Job Workers can also host the Balancer.

For each of the TarCloud jobs, a separate Azure application is implemented and stored in the BLOB storage. These applications are just thin clients who create HTTP (REST) requests. The requests tell the Balancer to occupy some of the Job Workers to execute a particular TarCloud job. Another VM, called Generic Worker (GW), has the responsibility to load the previous applications from the BLOB storage, and execute them in the order determined by the workflow of Figure 3.19, after a user request. User requests are accumulated into a job queue. To achieve synchronisation, GW just checks if the input for each job is ready. In brief, for each job, GW knows the URIs of its input files (in the BLOB storage) and, before the job execution, performs polling to check if the input files are complete. The execution starts only when the input files are ready. For instance, for the workflow of Figure 3.19, GW will not execute J31, until the output files of J1 (which are input for J31) are ready.

3.2.1.2 User interface and evaluation

We also implemented a Web interface, to give to TarCloud users an easy and intuitive way to start miRNA target prediction tasks. Figure 3.21 illustrates a snapshot of this interface. The user just selects the desired species from a drop-down list. Then, she selects one or more miRNA sequences. She has 3 options: (a) select already known miRNA sequences by inserting their names in the “Select miRNA by name” field (we store all miRNAs in the latest version of miRBase), (b) select unknown miRNAs by determining their sequences in the “Select miRNA by sequence” field, or (c) select a set of unknown miRNAs by uploading a file containing a name and a sequence for each of them. When the user clicks on the “Predict!” button, a request for the previously determined job is sent to the Generic Worker.

When the output is ready and stored in the BLOB storage, the TarCloud Web interface renders a webpage containing the output. The user has two options: (a) browse the predicted targets by using our build-in results renderer (see Figure 3.22), or (b) download the output text files to her own computer.

We also conducted experiments to evaluate the performance of our system by requesting predictions for 65 miRNA sequences. These sequences where human or mouse miRNA sequences randomly selected from miRBase (the database of all known miRNAs). Our infrastructure consisted of 3 Job Worker instances (native Azure medium-sized instances) and 1 Generic Worker instance. The results were verified by comparison to the output of the original DIANA microT implementation for the same miRNA sequences and are shown in Table 3.5.

6Azure applications are 32-bit Windows executables.
3.2.2 MR-microT: target prediction using MapReduce

3.2.2.1 Motivation

TarCloud inserts only trivial parallelisation to the prediction of the targets for a given miRNA. In particular, the sequence alignment step is broken into two disjoint sets of computations while the conservation profiling is broken into only a handfull of disjoint computation sets (based on the number of chromosomes). As a result, the execution time for a single miRNA cannot be significantly accelerated and thus cannot be scaled up by assigning more processing nodes. Moreover, the TarCloud implementation is dependent on the Azure platform making it impossible to port the system to another Cloud provider or to a private cluster if needed.

For the aforementioned reasons we introduced the MR-microT system [40], which is a MapReduce-based adaptation of the microT method. The key feature of MR-microT is the parallelism of the prediction process, so that the execution time for a single miRNA can be accelerated as desired by allocating more resources. This is achieved by carefully dividing the input data (genome and cross-species conservation information) among different processing nodes. This approach also achieves


<table>
<thead>
<tr>
<th>System</th>
<th>Average execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial implementation</td>
<td>37 min</td>
</tr>
<tr>
<td>TarCloud</td>
<td>5 min</td>
</tr>
</tbody>
</table>

Table 3.5: Measurements for TarCloud performance.

accelerating the execution for multiple miRNAs. Finally, MR-microT comes with an intuitive and powerful Web interface which can be used by researchers to produce target predictions for arbitrary miRNA sequences. As a result, MR-microT is the first microT implementation capable of near-real time prediction of ad hoc miRNA targets. Since, several design choices in MR-microT could apply to other similar computationally intensive bioinformatics methods, our ambition for MR-microT is to serve as an exemplary system in the field.

3.2.2.2 Technologies and system architecture

MR-microT is a distributed implementation of the microT target prediction method designed to run on computational clusters of arbitrary sizes. It utilizes the Hadoop framework for the distribution of microT computations in the nodes of the cluster. Hadoop is an open source framework that implements the MapReduce distributed programming paradigm [18] and can be easily installed in almost any contemporary computational cluster. The execution of a MapReduce program consists of the Map and the Reduce phase. During the Map phase, the input is split into disjoint pieces and each piece is independently processed by a different node of the cluster to produce a set of key-value pairs. The Reduce phase groups these key-value pairs by key and processes their values together to produce the output of the program. We use the Hadoop streaming utility to exploit large amounts of existing Perl scripts written by bioinformaticians.

Most of the data required by the microT execution are stored in the Hadoop Distributed File System (HDFS). This distributed file system lies on the hard disks of the nodes of the cluster and it is used for the storage of the input and the output of any Hadoop code. We use HBase, a non-relational distributed database inspired by Google’s BigTable [14], to store data that must be retrieved ad hoc during run-time.

The cluster which hosts MR-microT is a Virtual Network consisting of 19 Ubuntu Virtual Machines, provided by the Cloud service of ~okeanos [46]. ~okeanos is an IaaS platform providing resources to the Greek research and academic community.

The architecture of MR-microT is depicted on Figure 3.23. In particular, MR-microT consists of (a) a Web server, that collects user requests for target predictions of miRNA sequences, (b) a Hadoop cluster, on the nodes of which our code is executed, and (c) a Hadoop master, which is a Virtual Machine responsible for managing the Hadoop cluster and the HDFS and HBase storage resources.

The Web server is an Ubuntu Virtual Machine carrying an instance of the Apache server and holding MR-microT’s user interface (see Chapter 3.2.2.3). This front-end

---

10 [https://hbase.apache.org/](https://hbase.apache.org/)
11 [https://okeanos.grnet.gr/](https://okeanos.grnet.gr/)
is written in PHP and collects user’s HTTP requests to the MR-microT system for given miRNA sequences. The Web server sends these requests to the Hadoop master, which distributes the computations required by the requests to nodes of the Hadoop cluster. Each node reads input files from the HDFS during the initialization phase and requests HBase records when needed. Note that the master node is also the proxy of the HDFS and one of the servers of the HBase requests.

As the cluster nodes execute their tasks, the master monitors their progress. The Web server polls the master for progress reports and presents them to the user in a human-friendly form. The last progress report of the master informs the Web server that the process is completed, and, then, the output data are transferred from the HDFS to the Web server in order to be presented to the user.

There are three basic execution steps in the microT method for finding the targets of a miRNA molecule for a particular species. First, a set of candidate targets is produced by aligning the first 9 base pairs of the miRNA sequence inside the protein coding and the 3’-UTR region of each gene sequence of this species. A gene is called a candidate target if it contains at least one location in its protein coding or 3’-UTR region where these 9 base pairs can be aligned. These locations are called miRNA recognition elements (MREs) and they are candidate binding sites for the miRNA molecule inside the target. For each MRE, the quality of the alignment is recorded as its alignment score. In the second step, a conservation score is calculated for each MRE based on the number of species in which the MRE is preserved. Finally, in the third step, an aggregated score is calculated for each gene of the species, taking into consideration the computed scores for all its MREs, as well as some other factors, such as the folding of the involved molecules.

Our MapReduce design is based on the following observation. It is possible to perform the computations involved in the first two steps of the microT method independently for each gene. Therefore we choose to map these independent tasks to different nodes of the cluster (of course, if the total number of tasks is greater than the number of available nodes, then some nodes may need to execute consecutively multiple tasks). Such a task distribution can be realized by implementing the first two steps of the microT method as the Map phase of a Hadoop code. The input required in this Map phase is stored in the HDFS as a set of records, where each one contains data related to a region (protein coding or 3’-UTR) of a particular gene. The output of the Map phase would be a set of records as well, where each record encodes the location of an MRE, the scores calculated for it, and the identifier of the target, i.e., the gene, into which it resides.
The third step of microT consists of calculating a final score for each target by combining the alignment and conservation scores of all the MREs residing in it. Therefore, in MR-microT, this step is implemented as the Reduce phase, consuming the output of the previously described Map phase. For all records having the same target identifier, the Reduce phase calculates the final score considering also the molecular folding. More details on both the Map and Reduce phase of MR-microT follow.

The Map phase gets the protein coding or the 3'-UTR region of a gene sequence along with some information related to its preservation in a predefined set of species, and produces the MREs of that sequence along with their alignment and conservation scores. The input is organized in files stored in HDFS. Each line of these files corresponds to one (protein coding or 3'-UTR) region of a gene and it is structured as a key-value pair. The key of the pair is the concatenation of gene's Ensembl identifier with a string that denotes the type of the region (protein coding or 3'-UTR). The value of the pair is a struct containing the sequence of the gene region along with some basic information about the gene and its conservation in a number of species.

For each key-value pair, the Map code first executes a sequence alignment algorithm to find all the alignments of the first 9 base pairs of the miRNA sequence in the gene region sequence contained in the value of the pair. Each found alignment of these base pairs is an MRE and an alignment score is calculated for it based on the quality of the alignment and the strength of the bonds which are going to be created in case of a molecular binding.

Then, the conservation information of the gene sequence is considered and a conservation score is calculated. The more species that preserve the gene sequence unaltered exist, the larger the resulting conservation score is. Note that, the calculation of this score depends on precomputed weights for each possible pair of 3-grams (a pair consists of a gram from the reference species and a gram from another species). These weights are required ad hoc during the Map phase execution and are thus stored in HBase.

Finally, a set of key-value pairs, one for each found MRE, is produced as an output of the Map phase. The key of each pair is the Ensembl identifier of the gene in which the MRE resides and the value encodes the location of an MRE along with its calculated alignment and conservation scores.

The Reduce phase consumes the output of the Map phase. The Hadoop framework ensures that all the key-value pairs having the same key (i.e., the same gene identifier) are going to be processed by the same cluster node. This node aggregates the alignment and the conservation scores of all MREs found for this gene, and derives a prediction score for the gene itself. During this calculation, the Reduce phase also considers the folding of the involved molecules (this is important as the folding could actually destroy a possible binding). Note that the folding information is stored in HBase and is requested ad hoc during the execution. The output of the Reduce phase is recorded in HDFS and the Hadoop master is informed about its storage location.

3.2.2.3 User interface and evaluation

The services of MR-microT are accessible through a powerful yet intuitive Web interface. A screenshot of the interface is presented in Figure 3.24. First, the
user selects the species in which she wants to find targets from a drop down list. Currently, there are two available choices: Homo Sapiens (i.e., human) and Mus Musculus (i.e., mouse). Then, she inserts one or more miRNA sequences in a text box.

![MR-microT](image)

Figure 3.24: Screenshot of MR-microT’s user interface

When the species and the miRNA sequences are specified, the user can send his prediction request to the system. This is done by clicking the button labeled “Predict!” After that, the user has the opportunity to monitor the target prediction progress for each of the sequences she had specified. The system displays a separate progress view for each miRNA sequence into which the progress of target prediction is rendered. An example progress view is shown in the upper part of Figure 3.25.

When the target prediction method for a miRNA sequence is completed, the progress view is augmented with the result view, shown in the lower part of Figure 3.25, which contains information about the computed targets. Since the number of targets is usually large, they are organized into pages.

Each page displays information related to a number of targets. The information for target is rendered within the grey boxes shown in Figure ???. This information box contains the gene identifier and the final target prediction score calculated for this gene. Details regarding the list of the predicted binding sites of the target can be found by clicking on the arrow button located at the right of the grey box.

We performed some experiments to evaluate the performance of MR-microT. Table 3.6 summarises the average execution time required by MR-microT to produce the targets of a given miRNA in the human and mouse genome. For both measurements, we have tested three different cluster sizes, one containing 19 nodes, another containing 10 nodes, and a last one consisting of one node. Each node is a VM provided by Ωkéanos and containing two cores.

It is evident that the execution time per miRNA drops linearly as the size of the cluster increases. This is the main difference to TarCloud, which cannot reduce
the execution time per miRNA lower than 5 minutes by adding more computational nodes. In contrast, MR-microT could achieve even smaller execution time if we would deploy more resources. Of course there is a lower limit, because we cannot divide the input of each microT process in smaller pieces than the size of the genes, however, it is possible to produce target predictions for a given miRNA in less than a minute by executing MR-microT on a cluster of proper size.

Regarding the scenario of multiple, simultaneous user requests for the prediction of many miRNAs, MR-microT can serve all these requests with no observable delays, provided that its assigned resources (i.e., computational cores) are not exhausted. Otherwise, delays will be observed, however, eventually, the system will provide result to all of them. Note that by utilising services for elastic deployment of resources it is possible to avoid delays like the aforementioned.

### 3.3 Conclusions

Without efficient and accurate computational methods to predict miRNA targets, the required time and cost to perform biochemical experiments that reveal the role of particular miRNAs in the development and the treatment of important diseases would be huge. We selected to boost DIANA microT one of the most popular and accurate miRNA target prediction methods. Our goal was to achieve near real time production of the targets of a given miRNA.

In this direction, first we studied the sequence matching process that consists the
first step of the method. We found that it incorporates a novel sequence matching query. We formalised the aforementioned query introducing the ARSM problem. The objective of this problem is to retrieve all regional occurrences of a pattern in a data sequence. The matching regions of the pattern must contain a predetermined area of the pattern, the core. Moreover, the allowable deviation from the data sequence is stricter for smaller and looser for larger regions. To deal with the previous problem, we proposed \textbf{PS-ARSM} method. Our method takes advantage of the prefix and suffix overlaps avoiding redundant computations. A detailed experimental evaluation showed that \textbf{PS-ARSM} is up to two orders of magnitude faster than existing techniques adapted to the ARSM problem.

However, accelerating the sequence matching step of target prediction methods is not enough to achieve near-real time performance. This is because these methods also incorporate some other computational intensive processes (e.g., computing the folding of the involved molecules, considering the way any predicted binding site is preserved in a multitude of species, etc). We followed the approach to distribute these computations in the nodes of a Cloud infrastructure. In this direction, we designed two Cloud-based target prediction systems, TarCloud and MR-microT. The former was developed using the framework of Microsoft Azure, while the later using the Hadoop framework. Based on our measurements both systems accelerate the prediction process, however MR-microT is superior because it is platform-independent (it can be installed in any cluster of contemporary computational nodes), it provides improved parallelisation of the involved tasks, and it is designed to support without problems increased number of prediction requests.
Chapter 4

Research infrastructures for miRNA research

In this chapter, we present a set of infrastructures developed to support miRNA research. Until recently, important information about the function and the regulation of each miRNA was scattered in many databases or even not available at all. This was an important obstacle for researchers in life sciences who were trying to understand the role of miRNAs in many biological pathways, a knowledge that could help towards discovering treatments for particular diseases. In Chapter 4.1 we record some interesting data and tools that could be valuable for the life scientists in the field of miRNA research. The list was made by questioning experts of the field.

Based on the recorded needs, and in collaboration with a research team from BSRC Al. Fleming, we developed DIANA tools\(^1\), a large set of publicly available databases and Web tools dedicated to facilitate research in the area of life sciences that is related to miRNAs. In the remainder of this chapter we present in detail one by one all these tools discussing their motivation, their functionality and their contributions.

4.1 Needs for miRNA tools and data

There is a plethora of repositories that collect and distribute interesting information of general interest in life sciences. An indicative example is the set of resources provided by Ensembl and NCBI. Nevertheless, there is an absence of similar resources in the field of miRNA research. Although miRBase\(^2\) is a hub collecting interesting information related to each existing miRNA molecule, many useful data regarding miRNAs are either scattered in scientific publications or not existent at all (since additional analysis should be performed to reveal them). In this chapter, based on questioning experts of the field, we collect and record some miRNA related data that belong to the aforementioned category.

The most important information about a miRNA molecule is the list of genes affected by its presence, i.e., the list of its target genes. A target gene can be revealed by biochemical experiments. The results of such experiments are published in scientific journals. Therefore, a researcher interested in the targets of a particular

\(^1\)http://http://diana.imis.athena-innovation.gr
\(^2\)http://www.mirbase.org
miRNA should search in the whole available bibliography to find the experiments of his interest. Some curated databases collecting the miRNA-gene interactions exist, however they contain very limited number of interactions and are not properly updated.

It is expected that there exist millions miRNA-gene interactions. However, the already performed biochemical experiments have revealed only a small part of them. This is because a set of experiments like the aforementioned require significant amount of time to be completed. This is why, computational methods that rapidly predict miRNA-gene interactions are very valuable since they provide insights about possible relationships between miRNAs and genes. There is a multitude of prediction methods that have been proposed. Some of them provide a Web interface presenting the predicted targets of all known miRNAs. DIANA microT is one of the most accurate target prediction methods, however, it lacks in disseminating its predictions as it does not provide any Web interface to present them.

Another important information for miRNAs is the exact genomic location of their transcripts and their expression profiles. Again this information is scattered in relevant scientific publications. The interaction of miRNA transcripts with particular transcription factors, the list of tissues into which these transcripts are expressed, and the metabolic pathways into which they participate could also be powerful pieces of information for the life scientists in the field of miRNA research.

Finally, since there is a huge number of miRNA-related publications, which are constantly increasing, the need to search for all publications that are relevant to a particular molecule arises. This is not a trivial task since miRNA nomenclature is used inconsistently in the literature and even the official miRNA names can change from time to time.

Our goal is to develop a set of powerful tools that will cover the information gap existing in the above mentioned areas. In Chapter 4.2 we present several versions of DIANA microT, a Web server providing predictions for the genes that are targeted by all known miRNAs, based on the very accurate DIANA microT prediction method. In Chapter 4.3 we describe DIANA miRGen, a system that informs its users about the genomic locations of all miRNA transcripts and their expression behaviour. Moreover, in Chapter 4.4 we introduce DIANA TarBase, a database collecting experimentally verified miRNA targets. In Chapter 4.5, we present DIANA mirPath, a tool that investigates the role of miRNAs in the known metabolic pathways. Finally, in Chapter 4.6, we discuss DIANA mirPub, a Web tool assisting life scientists in miRNA-related literature search. We believe that the aforementioned tools is a valuable addition in the toolbox of any life scientist in the area of miRNA research.

4.2 DIANA microT Web server: Searching for miRNA predicted targets

In the decade of 2000, a large number of miRNA target prediction methods have been introduced [87, 2]. Most of them were mainly based on sequence alignment of the miRNA seed region (i.e., nucleotides 2−7 from the 5'-end of the molecule) to the 3'-UTR of candidate target genes to identify putative binding sites. Moreover, to improve their precision, these methods used to exploit some additional features such
as the evolutionary conservation of the candidate binding sites, structural availability of the involved molecules, nucleotide composition, etc. For more details on the techniques used by miRNA target prediction methods see also Chapter 2.2.2.

This is when the first version of DIANA microT method was introduced [43]. In 2009, the research team that was responsible for the development of DIANA microT was working on the third version of the method. They established a partnership with us and, in this framework, we started working on (a) techniques to improve the efficiency of their method and (b) research infrastructures to propagate its results. The first output of this cooperation was the DIANA microT v.3 Web server [59], a research infrastructure that collects the predicted targets of all known miRNAs in the genomes of human and mouse based on DIANA microT method. This Web server became very popular (counting more than 150 unique visitors per day) because of the accuracy of DIANA microT v.3 method and the great variety of features provided through its well designed Web interface. During the subsequent years, two more versions of DIANA microT Web server were released [60, 76] becoming even more popular (more than 500 unique visitors per day). In the following paragraphs we discuss the contribution of the aforementioned versions of DIANA microT Web server.

4.2.1 DIANA microT v.3

DIANA microT v.3 method identifies locations in the 3'-UTR region of the genome that possibly correspond to a miRNA binding site based on sequence alignment. Then, for each of these predicted sites, a score is calculated based on the quality of the alignment, the conservation of the site in a multitude of species, and the folding of the involved molecules. The scores of all the predicted sites of each gene are aggregated to produce the miTG score, an indicator of the likelihood that the miRNA targets the gene for real. For more details about the method see also Chapter 2.2.2.1.

DIANA microT v.3 Web server [58, 59] was developed to collect the predicted target sites of all known miRNAs in the human and mouse genes and, then, propagate this knowledge to the life sciences researchers, through a powerful and intuitive Web interface. The user can search for the predictions related to a particular miRNA, gene or biological pathway by inserting keywords that describe these entities in search boxes. Each predicted miRNA-gene interaction besides the miTG score, comes with a signal-to-ratio (SNP) and a precision score. These scores can be used as indicators about the correctness of each prediction. Moreover, each predicted interaction is presented along with links to external biological resources and to related publications. Interactions predicted by other prediction methods or interactions verified by biochemical experiments are marked properly in the interface. Finally, the server is integrated in platform with two further miRNA functional analysis tools: mirPath, a pathway analysis tool of predicted targets and mirExTra, a miRNA analysis based on differential expressed mRNA profiles. Figure 4.1 presents a snapshot of the DIANA microT v.3 results page for the combined search of a miRNA (“hsa-let-7a”) and a gene (“CCND1”).

DIANA microT v.3 also supports prediction requests for user-defined miRNA sequences. The results of the de novo predictions are stored in a database from KEGG (Kyoto Encyclopedia of Genes and Genomes) pathways were considered [37].
Figure 4.1: A screenshot of DIANA microT v.3 interface while showing the predicted targets of miRNA “hsa-let-7a” in the gene “CCND1”.

which they can later be retrieved and presented to the user who is provided with a unique key via email notification. This scheme was necessary because, at that time, de novo prediction for a miRNA sequence was computationally intensive requiring more than 20 minutes to be completed on a cluster of 32 nodes (256 cores) hosted at National Technical University of Athens (NTUA). Recall that Cloud approaches were used to achieve near real time de novo predictions (see Chapters ?? and 3.2.2).

Note that all DIANA microT v.3 Web server user interfaces were developed using PHP, while all the required data were stored in a relational database (MySQL was used). NTUA was hosting the Web tool at http://diana.cslab.ece.ntua.gr/microT/.

4.2.2 DIANA microT v.4

DIANA microT v.4 Web server [60] is an extensive update of DIANA microT v.3 with several important improvements:

- Correlation of miRNAs to diseases based on an advanced bibliographic analysis
- Support of predictions for two additional species (Drosophila melanogaster and C. elegans)

- Graphical display with all relevant functional information from the UCSC genome browser,

- Support to old miRNA names by tracking changes in miRNA nomenclature

- User personalised sessions allowing personal query history and bookmarks

DIANA microT v.4 Web server provides functional analysis of miRNAs that reaches beyond a simple listing of miRNA targets. This is achieved through the integration of knowledge extracted both from scientific literature and the information included in known biological pathways. Towards this direction, the Web server provides associations of miRNAs to diseases, based on trivial text analysis performed on titles and abstracts of PubMed publications. In particular, a miRNA was considered to be related to a disease if there is at least one publication that (a) contains the miRNA name or its family name in its title or abstract and (b) is annotated with a MeSH term describing the disease. All the MeSH diseases found to be related to a miRNA are visualised through a tag cloud that helps user to get an insight about the knowledge described in the literature. The MeSH terms in the tag cloud also serve as hyperlinks to the relevant publications.

Another important new feature of DIANA microT v.4 Web server is that it collects predicted targets for two more species. In particular, the first version of the server was designed to support the functional analysis of human and mouse miRNAs. DIANA microT v.4 server has been updated to contain also predictions for Drosophila melanogaster and C. elegans. Moreover, data from more recent versions of miRBase (miRBase 13) and Ensembl (Ensembl 54) were included in the database of the new Web server. In total predictions for 723 new miRNAs have been added, out of which 147 correspond to Drosophila melanogaster and 154 to C. elegans (the rest are new Homo sapiens and Mus musculus miRNAs). This results in an approximately doubled number of predicted targets in comparison to DIANA microT v.3, counting more than six million predicted target genes. Finally, while DIANA microT v.3 is based on features separating real and mock (shuffled) miRNAs (see also Chapter 2.2.2.1) DIANA microT v.4 uses high-throughput experimental data for the same purpose.

The interface of DIANA microT Web server was completely redesigned and became more intuitive (see Figure 4.2). The most important change was that instead of providing individual search boxes to search for miRNAs, genes and pathways, the new Web server provides only one search box supporting all types of search. In addition, the positions of the binding sites on the transcript of the target gene are graphically presented through the UCSC genome browser. This automatic upload can be used to provide information in comparison to other tracks of interest such as single nucleotide polymorphisms (SNPs), repeat elements, and alternative 3'-UTR splice forms. Another important improvement is that DIANA microT v.4 provides an integrated personal user space in which users can easily save important searches and results that they wish to keep for future analysis. In particular, the system keeps the most recent user searches, giving the opportunity to repeat searches. A

---

4 Medical Subject Headings provided by the National Library of Medicine.
bookmarking mechanism provides the opportunity to save interesting results along with user comments. The personal space provides usage statistics regarding the most recent searches, thus, enabling them to keep track of their latest findings.

Since miRNA biology is still a field in flux, it can occur that a miRNA may change name between two successive versions of miRBase. Due to such changes, researchers may lose track of a miRNA full history and related literature searches will remain incomplete. To address this issue, an extended analysis on 13 versions of miRBase (versions 7.1. to 14) is performed, and the nomenclature history of each miRNA is extracted. The analysis uses version 13 of miRBase as the reference database. This version includes 1,884 mature miRNAs for the four species supported in the Web server. Each miRNA is assigned a unique identification number denoted as ‘MIMAT id’ and one associated miRNA specific name. Among versions, changes are found in 77 MIMAT ids (38 in human, 37 in mouse and 2 in Drosophila) and 151 miRNA names (76 in human, 71 in mouse and 4 in Drosophila). This indicates that name changes are more frequent than changes in MIMAT ids. To keep track of these changes, a history index was integrated in the Web server. This index information was made available to the user through a specific feature called ‘miRNA history’ which was also used for miRNA related bibliography searches (see Figure 4.2). For example, miRNA “mmu-miR-455” first appeared in miRBase v8.1. Its name was later changed to “mmu-miR-455-5p” in version 8.2 and later appeared as “mmu-miR-455*” in version 10.0 (see Figure 4.3).

Note that all DIANA microT v.4 Web server user interfaces were implemented in PHP using well established design patterns for Web development (like MVC model, Active Records, etc). All the required data were stored in a relational database.
Figure 4.3: The data evolution history related to miRNA with name “mmu-miR-455*”.

(MySQL was used). The service ViMa provided by GRNET was used to host the Web tool at http://diana.imis.athena-innovation.gr/DianaTools/index.php?r=microtv4.

4.2.3 DIANA microT v.5

DIANA microT v.5 Web server [76] is a significantly updated version of DIANA microT v.4 Web server. It utilizes the state-of-the-art target prediction method DIANA microT-CDS [78], which is specifically designed to identify miRNA targets both in 3'-UTR and the coding sequences of genes (for details see also Chapter 2.2.2.3). This is very important since, although initial research was indicating that miRNAs bind only on the 3'-UTR regions of genes, accumulated experimental evidence has revealed that miRNA-binding sites within coding sequences are also functional in controlling gene expression [92]. Moreover, DIANA microT CDS method provides increased accuracy and the highest sensitivity at any level of specificity over other available state-of-the-art implementations, when tested against pulsed stable isotope labeling by amino acids in cell culture (pSILAC) proteomics data sets [83].

DIANA microT v.5 server hosts miRNA targets in gene sequences of Homo sapiens, Mus musculus, Drosophila melanogaster and C. elegans. In particular, its database, during the date of its initial release, was containing $7.3 \times 10^6$ interactions for H. sapiens, $3.5 \times 10^6$ for M. musculus, $4.4 \times 10^5$ for D. melanogaster and $2.5 \times 10^5$ for C. elegans interactions. Furthermore, DIANA microT v.5 hosts, in comparison to DIANA microT v.4, updated information about miRNAs (miRBase v.18), genes (Ensembl v.69), and KEGG pathways [38]. Finally, gene and miRNA expression annotation has been incorporated into the web server, enabling the user to perform advanced result filtering based on tissue expression.

Besides the basic search functionality, the fifth version of the DIANA microT server provides also support for advanced high-throughput analysis pipelines. In particular, DIANA microT v.5 Web server hosts numerous integrated analyses in the form of ready-made advanced pipelines, covering a wide range of inquiries regarding predicted or validated miRNA-gene interactions and their impact on metabolic and signalling pathways. These pipelines can be used to analyse user data derived from small scale and high-throughput experiments directly from the DIANA-microT web server interface, without the necessity to install or implement any kind of software.
For instance, one of the available workflows (Figure 4.4) can analyse mRNA and miRNA expression data (expression and fold change). Suppressed genes are automatically matched with overexpressed miRNAs (and vice versa). The workflow performs enrichment analysis of experimentally validated targets derived from DIANA TarBase v.6 (see Chapter 4.4) or/predicted interactions from microT-CDS. This step is considered crucial to identify miRNAs that significantly regulate the differentially expressed genes.

![Flowchart depicting an analysis pipeline directly available from the Web server interface.](image)

The prediction score threshold can significantly affect the analysis steps that follow. In the case of predicted interactions, the pipeline can be optimized by automatic repetitions of different prediction thresholds (from sensitive to more stringent), to minimize the effect of the selected settings to the result. By using meta-analysis statistics, the server combines the P-values from each repetition into a total P-value for each miRNA, signifying its effect on the selected genes for all used thresholds. In the last step of the pipeline, the identified miRNAs are subjected to a functional analysis, where pathways controlled by the combined action of these miRNAs are detected using DIANA mirPath v.2.1 (see Chapter 4.5).

Other available pipelines can handle miRNA and gene lists, to perform the enrichment analysis, or even select the type of used interactions (predicted or experimentally validated). In the latter workflow, the algorithm ‘personalizes’ the target identification module for each miRNA. It initially identifies the number of available interactions in DIANA TarBase and DIANA microT-CDS (validated versus predicted) and automatically selects to use validated targets only in the cases of well-annotated miRNAs. Computationally identified interactions are used otherwise.

The new DIANA microT Web server enables users to perform such analyses directly from the on-line user interface, or create even more extensive pipelines programmatically or by using visual tools (Taverna WMS [33]). To this end, DIANA microT Web server v5.0 provides a complete integration with the Taverna WMS, using our in-house developed DIANA-Taverna Plug-in. DIANA-Taverna Plug-in en-
ables the user to directly access our target prediction server (microT-CDS) from the graphic interface of Taverna and incorporate advanced miRNA analysis functionalities into custom pipelines. Furthermore, the plug-in enables the extension of such pipelines through the use of other DIANA tools and databases, providing access to the most extensive collection of validated miRNA targets (DIANA TarBase v.6) and to DIANA mirPath v.2.1, a tool designed for the identification of miRNA targeted pathways. Furthermore, the Web server also supports direct programmatic access to all aforementioned utilities in the form of services, to facilitate users having already implemented pipelines using scripting or programming languages.

Note that all DIANA microT v.5 Web server user interfaces were implemented in PHP using well established design patterns for Web development (like MVC model, Active Records, etc). All the required data were stored in a relational database (MySQL was used). The service ViMa\(^5\) provided by GRNET\(^6\) was used to host the Web tool at [http://diana.imis.athena-innovation.gr/DianaTools/index.php?r=microtv5](http://diana.imis.athena-innovation.gr/DianaTools/index.php?r=microtv5).

### 4.3 DIANA miRGen: Revealing information about miRNA transcripts

Like other RNA molecules, mature microRNAs are produced by transcripts. Each of these transcripts can encode more than one mature miRNA (see also Chapter 2.2.1.3). Knowing the genomic location of these transcripts and when they are expressed is a very useful information for any researcher studying miRNA molecules. The first version of DIANA miRGen \[63\] was a database capturing the aforementioned information. However clusters of miRNAs that are expressed together were identified based on their relative distance and the genomic features surrounding them. This methodology results in many errors, thus, a more accurate database was required.

#### 4.3.1 DIANA miRGen v.2

DIANA miRGen v.2 \[3\] is a database that aims to provide comprehensive information about the position of human and mouse microRNA coding transcripts and their regulation by transcription factors, including a unique compilation of both predicted and experimentally supported data.

During the development of DIANA miRGen v.2:

- primary transcripts in mammalian genomes (in particular, in the human and mouse genome) were identified by mining prominent literature sources.

- transcription factors binding sites (TFBSs) were mapped within the regions upstream of the transcription start sites (TSSs) of the aforementioned miRNA primary transcripts

- expression profiles of miRNAs in several tissues, the mapping of SNPs within genomic locations of miRNA hairpins, and the mapping of SNPs within the TFBSs found upstream of miRNA genes were incorporated.

\(^5\)http://vima.grnet.gr
\(^6\)https://www.grnet.gr
The interplay of these different information sources concerning genomic features associated with miRNA genes and their expression levels can be used to study the function of miRNAs and their deregulation in disease. For instance, a user interested in a specific transcription factor can find miRNA genes associated with it, find the expression levels of these miRNAs in a possible tissue of interest, possibly find some SNPs on the TFBSs or the miRNA locations on the genome that relate to a possible disease of interest and, finally, find predicted targets of the miRNAs associated with the transcription factor of interest, and molecular pathways in which the targets of each of these miRNAs separately or together are implicated.

MiRNA transcripts in human and mouse were identified from four literature sources:

- Corcoran et al. [17] used PolIII immunoprecipitation data and ChIP-chip on lung epithelial cells to identify miRNA transcripts and their promoter regions.

- Landgraf et al. [48] sequenced 250 small RNA libraries corresponding to 26 different organ systems and cell types of human and mouse, with approx. 1,000 miRNA clones per library and identified miRNA coding genes. In this study the whole transcripts of miRNA coding genes were identified, as well as protein coding genes that contain miRNAs.

- Oszolak et al. [72] predicted the location of the proximal promoters of human miRNAs by combining nucleosome mapping with promoter chromatin signatures in MALME, HeLa and UACC62 cells. Although the TSS of miRNA genes was identified in this study, the end of the transcript was not provided. We have provided end of the last miRNA that is a member of a gene as an approximation of the transcript end.

- Marson et al. [61] used ChIP-seq data to identify promoters of miRNA genes in embryonic stem cells. They identified promoters and co-regulated miRNAs, but the exact position of the TSS was not identified. For this reason we have used the start of the first miRNA of each cluster as the putative TSS. Additionally, coordinates provided by Marson et al. had to be lifted over using ‘UCSC lift over tool’ to the most recent, at that time, genome build (hg18, mm9). In cases where putative rather than experimentally verified positions are used, they are denoted in the graphical interface as ‘computational TSS’.

In total, 812 human miRNA coding transcripts and 386 mouse miRNA coding transcripts were identified. Of them, 423 were shown in the corresponding papers to be associated with protein coding genes (intragenic miRNA transcripts). More than one of the above publications have usually identified transcripts corresponding to a miRNA. When this is the case, transcripts from all methods are returned to the user.

Since these studies were published, additional miRNAs have been identified. When novel miRNAs are located within the coordinates of clusters given by any of these publications, this miRNA is added to the cluster. For names that changed or were given differently than the current standard, manual curation with reference to mirBase was used to identify and replace these names according to the current standard. For all the above reasons it is possible that the number of genes used in...
miRGen does not correspond perfectly to the number stated in the corresponding publications.

In order to determine putative TFBSs near the TSS of miRNA primary transcripts, we used the freely available tool MatchTM[41]. MatchTM uses the public library of position weight matrices from Transfac 6.0. We matched all vertebrate transcription factor matrices to the regions spanning from 5 kb upstream of each TSS to 1 kb downstream of the TSS. As criterion for determining the cut-off values we chose the minimisation of false positives in order to produce a strict set of predictions without too many falsely predicted TFBSs. Two scores are calculated for each putative TFBS. The matrix similarity score describes the quality of a match between a whole matrix and an arbitrary part of the input sequences. Analogously, the core similarity score denotes the quality of the match between the core sequence of a matrix (i.e. the five most conserved positions within a matrix) and a part of the input sequence.

miRNA expression profiles were identified from the mammalian miRNA expression atlas [48]. Information for the expression profiles of 548 human and 451 mouse miRNAs over 172 human and 68 mouse small RNA libraries were derived from cell lines and tissues.

SNPs located within the genomic positions of miRNA hairpins and corresponding TFBSs were downloaded from the UCSC table browser. For human, Polymorphism data from dbSup database [88] or genotyping arrays SNP130 were used with 18,833,531 identified SNPs. For mouse, SNP128 was used with 14,893,502 identified SNPs.

The miRGen repository has been implemented using relational database technology. All data are stored in a MySQL relational database management system. Figure 4.5 illustrates part of the entity-relationship model of our application. For more details about the biological entities described in this model and the relationships between them see Chapter 2.2.1.

![Figure 4.5: Part of the entity-relationship model of miRGen’s database](image)

All results are available through a user-friendly interface that allows searches
for miRNAs and for transcription factors of interest. For mature miRNAs, it is possible to view targets predicted by the method DIANA microT and for miRNAs found in the same transcript, the user can see a functional annotation of their targets on molecular pathways through the application DIANA mirPath (see Chapter 4.5). Figure 4.6 shows an overview of the interface and highlights links to external databases-UCSC genome browser, iHop, dbSNP, and mirBase.

Figure 4.6: Two screenshots of miRGen interface. One for a miRNA search (A) and the other for a transcription factor search (B).

When a miRNA search is performed (Figure 4.6-A), all distinct locations on the genome (hairpins) that could code for this miRNA are returned, and the user can see details for any of the possible overlapping transcripts identified for each location, usually predicted by different papers. Each transcript tab contains information about TFBSs located from 5 kb upstream to 1 kb downstream of the transcript start. Additionally, information on the expression levels of the mature miRNA are displayed as a heat map.

Searching for a transcription factor of interest (Figure 4.6-B) returns all miRNA coding genes for which at least one binding site for this transcription factor is found. Information on the gene, the TFBSs, and the mature miRNAs coded for by the gene can be seen in tabs. All instances of TFBSs and miRNA hairpins are associated with corresponding SNPs mapping on their genomic locations. For all transcripts, the literature source of the gene is displayed, the identification of the TSS (experimental if the TSS was identified in the paper, computational if it was calculated by computational means and first miRNA if the start of the first miRNA serves as a substitute for an unknown TSS), and whether the gene is intragenic or is co-expressed with a protein-coding gene.

Note that all DIANA miRGen v.2, user interfaces were implemented in PHP using well established design patterns for Web development (like MVC model, Active Records, etc). All the required data were stored in a relational database (MySQL was used). The service ViMa provided by GRNET was used to host the Web tool at http://diana.imis.athena-innovation.gr/DianaTools/index.php?r=mirgen.
4.4 DIANA TarBase: Searching for experimentally verified miRNA targets

In the past decade, vigorous research efforts have provided significant amounts of data related to miRNA biogenesis and function. This process is reflected in the super linear increase of records in miRBase and of miRNA-related articles in PubMed (see Figure 4.7). Under these circumstances miRNA target identification becomes a tedious task for life sciences researchers.

Figure 4.7: The annual growth of miRNA-related publications in PubMed and the number of entries in miRBase database.

A large number of target prediction methods, that attempt to tackle the problem computationally, are available (see Chapter 2.2.2). Some targets can be confidently predicted with currently available techniques, however, precision and recall of state-of-the-art algorithms were estimated as approx. 50% and 12%, respectively, when tested against proteomics supported miRNA targets [2], highlighting the necessity for mass experimental miRNA target validation [93].

miRNA targets can be experimentally verified with gene-specific, as well as, high-throughput techniques. Specific techniques include reporter gene assays, assessment of miRNA and target mRNA co-expression (e.g. northern blotting or qPCR) and estimation of miRNA effect on target protein (e.g. ELISA, western blotting, immunohistochemistry). High-throughput techniques can be a simple extension of an existing gene-specific technique in a high-throughput setting, for example the utilization of microarray screening instead of qPCR. They can also involve novel relevant methodologies, such as RNA-Seq, immunoprecipitation of RISC components, high-throughput sequencing of RNA isolated by crosslinking immunoprecipitation (HITS-CLIP), photoactivatable-ribonucleoside-enhanced crosslinking and immunoprecipitation (PAR-CLIP), biotin tagging of miRNAs, parallel analysis of RNA ends (PARE) and various proteomics approaches such as SILAC.

As the relevant literature and the number of experiments increase in a super linear fashion, databases that curate and collect experimentally verified targets have gradually emerged. These databases attempt to provide efficient access to this wealth of experimental data, which is scattered in thousands of manuscripts. DIANA TarBase v.1 [86], the first database for experimental supported miRNA targets was created to assist for the design of more robust prediction methods. As the accumulated data increased, DIANA TarBase and the other related databases that followed widened their scope of application and became invaluable tools to all
facets of miRNA-related research. In the remainder of this chapter first we present an overview of the aforementioned databases and then we describe DIANA TarBase v.6 which is our development.

4.4.1 Related work

A brief overview of the available databases is presented below in alphabetical order:

miR2Disease [35] was first released in 2008. It is a manually curated database that aims to provide information regarding miRNA-related pathologies. miR2Disease curates 809 miRNA-gene interactions for Homo sapiens, coupled with related disease information derived from relevant literature. The 3,273 miRNA disease-related entries consist the strongest point of the database. The user can search by miRNA, target gene or disease name. Further details include method of validation, validated targets from an earlier version of TarBase, relation with the pathology, manuscript information and links to target predicting algorithms.

MirnaMAP [31] was first released in 2006. It contains data derived from an outdated version of DIANA TarBase (346 targets) and by manual curation (29 targets). MirnaMAP has not been updated for more than 4 years and contains a limited amount of experimentally validated targets for H. sapiens. The largest amount of mirNAMAP entries is based on predicted interactions for 2,464 miRNAs in 12 species. MirnaMAP provides a wealth of available data for each database entry, including miRNA and gene information, bead-array microRNA tissue expression profile, qPCR tissue expression profile, predicted target genes, as well as relevant literature.

MiRecords [104] was first released in 2008. It contains manually curated and predicted miRNA targets. The validated targets component of the database contains 2,286 interactions between 548 miRNAs and 1,579 target genes in nine species (these numbers refer to the version available on 25 November 2010). The largest number of those interactions is derived from gene-specific experiments. The database provides miRNA, gene and target site-related information, as well as links to miRBase and RefSeq. miRNA-gene interactions are supported with data regarding manuscript information, experimental method used for validation, as well as a selected passage from the manuscript stating the experimental result. However, the user does not have the ability to filter results based on any of the available predicted or validated component fields. The miRecords interface also enables the user to insert new miRNA-target interactions.

miRSel [65] database was first released in 2010. It contains miRNA interaction data derived solely from text mining of MedLine abstracts. The text mining algorithm manages to extract miRNA-gene associations with 65% precision, 90% recall, based on a test performed on 89 selected sentences, derived from 50 PubMed abstracts. MiRSel contains 3,690 miRNA-gene text mined associations. By applying less stringent criteria, the user can have access to approximately 8,000 pairs, which are deemed as less reliable by the developers. In miRSel, the user can also search for miRNAs related to specific MedLine articles that contain a subset of desired terms or that are related to Gene Ontology entries. Links to external databases such as miRBase and Entrez Gene are provided for each entry. Information regarding the experimental method used for miRNA target validation is not available. Data derived from other curated miRNA interaction databases such as DIANA TarBase v.5,
miR2Disease and miRecords have also been integrated.

miRTarBase [32] was first released in 2010. It includes manually curated data for 3,969 experimentally verified miRNA-gene interactions for 14 species (numbers for the version available on 15 April 2011). It provides information related to the miRNA, the target gene and the target site. In many cases, where the articles do not explicitly present target site information, miRTarBase can provide predicted regions by using a computational target prediction algorithm. Information regarding available experimental findings supporting the interaction is also included. The user-interface provides links to external data sources such as NCBI Entrez, UCSC Genome Browser, miRBase, BioGPS, iHOP and HGNC. Optionally the user can submit data for non-indexed interactions.

miRWalk [20] was first released in 2010. It provides experimentally supported miRNA targets identified solely from text-mined abstracts available in MedLine. The miRWalk validated targets module hosts text-mined interactions for 1,572 miRNAs interacting with 5,080 genes for three species (human, mouse and rat). A direct estimation of the system’s accuracy in interaction extraction has not been provided. The text mining approach of the authors enabled them to also collect data for disease targets, organs, cell lines and pathways.

StarBase [106] was first released in 2010. It is a platform focused on the analysis of high-throughput CLIP-Seq (HITS-CLIP and PAR-CLIP) and degradome sequencing (Degradome-Seq and PARE) data for six organisms. StarBase miRNA-related data are derived from eight different studies. The developers utilized five prediction programs to locate putative targets, which were subsequently intersected with the previously analyzed high-throughput data, resulting in a high number of putative targets (approx. 500,000). The user can reduce the number of false positives by selecting the results of only one prediction algorithm. StarBase provides the in-house developed deepView Genome Browser, which enables access to mapped reads, predicted and known miRNA targets, ncRNAs, protein coding genes, target clusters, target peaks and target plots. Provided information includes miRNA and gene-related info, GO terms and KEGG pathways related to each target gene.

DIANA TarBase v.5 [74] was first released in 2005 and its fifth version includes 1,300 experimentally supported targets from eight species that were manually curated from relevant literature. The provided data for each interaction include miRNA information, target gene general information, the nature of the experimental validation, cell line, sufficiency of the site to induce translational repression and/or cleavage and the supporting manuscript. The database also incorporates functional links to external sources of data for each entry, such as Ensembl, HUGO, UVSV and Swiss-Prot.

### 4.4.2 DIANA TarBase v.6

As miRNA-related research advances, the requirements for relevant databases increase significantly. An evident necessity exists for databases that manually curate large numbers of experimentally validated miRNA targets from gene-specific as well as from high-throughput techniques. Databases containing both types of experimental data provided until today contain less than 5,000 validated targets, with a significant portion of those targets derived from a few high-throughput experiments. Incorporating a large number of supporting studies increases data validity and en-
ables such databases to become concise datasets for training and testing robust miRNA prediction algorithms. Another important issue which soon became evident was that researchers require advanced searching and result filtering capabilities, in order to accurately discover miRNAs or genes of specific interest. Finally, the enhancement of the datasets with added information from external sources (such as pathways or terms from GO ontology) and metadata can enable efficient data mining of the existent experimentally validated results, producing useful novel observations.

The aim of DIANA TarBase v.6 [99] is to face the aforementioned current challenges and to inaugurate the next generation of validated miRNA target databases by providing a significant increase of available targets derived from all available experimental techniques, while incorporating a powerful set of tools in a user-friendly interface.

Manual miRNA-related literature curation is a time consuming process that requires highly trained personnel. A text mining-assisted human curation pipeline was implemented, in order to reduce the necessary time for manuscript curation. Since accurate automatic association extraction from biomedical literature is still an open problem, the text mining application was designed at this point only to assist the human curators in their task. A straightforward pipeline was designed that included NER (named entity recognition), miRNA-target association identification, scoring and enhanced text presentation (Figure 4.8). The designed module provides: markup of genes, miRNAs and associating words with different notations. Subsequently, scoring and sorting of abstracts are performed based on a target existence probability score. Specific markup for sections of increased interest is finally added; for example by underlining sentences that seemed highly probable to include a miRNA-target association.

![Figure 4.8: The adopted text-mining-assisted curation pipeline.](image)

Initially, all relevant abstracts containing miRNA-related terms (microRNA or micro-RNA or miRNA or ‘micro RNA’) in their title, text, keywords or MesH terms were downloaded from MedLine in the form of XML texts. The subsequent named entity identification process was divided into two distinct steps: in the first step, genes were identified using AIIA Lab’s AIIAGMT, one of the highest ranking named entity recognition systems in BioCreative challenges [89]. AIIAGMT offers an XML-RPC interface through a Perl module, implemented by the authors, which was incorporated into our pipeline. Our script marked specifically all mentions for genes discovered by AIIAGMT. The recognition of miRNAs is much more straightforward, since they follow a much more conservative nomenclature and do not possess a high number of aliases, which is common ground in gene notation. MiRNA recognition was therefore performed with regular expressions implemented in Perl. Following
NER, a list of 16-word stems was used as a basis for associative word recognition within the text. These stems were derived by removing suffixes (from verbs or nouns) from a superset of miRNA-target association denoting words (e.g. target-s, targeting, target-ed, etc.). Finally, a scoring function was applied that promoted abstracts containing a large number of sentences containing miRNA-target associations. This tool significantly increased daily curation outcome and enabled us to incorporate an unprecedented amount of curated targets in DIANA TarBase.

A new relational database schema was designed and implemented to accommodate present and planned future DIANA TarBase updates. DIANA TarBase v.6 hosts a significant amount of information for each miRNA-gene interaction ranging from miRNA and gene-related facts, to information specific to their interaction, the experimental validation methodologies and their subsequent outcomes. All database entries are enriched with a significant amount of function-related data, as well as general information derived from external databases. Database entries are mapped to external sources such as UniProt, Ensembl, RefSeq and others, in order to provide seamless integration with other services. DIANA microT v.4 [60] miRNA target prediction scores and links to the relevant microT entry have also been added to the interactions. The new extended database schema was designed to accommodate miRNA-gene interaction data in high detail and to be efficient during query evaluation. The database is supported by a large number of indices and materialized views for performance enhancement.

For each entry in DIANA TarBase v.6 the curators noted the miRNA, the related target gene as well as information regarding the experiment such as the utilized cell line or tissue. The utilised methodology (gene specific or high throughput) was specifically mentioned. The supported methodologies were ‘Reporter genes’, ‘qPCR’, ‘Western blotting’, ‘MicroArray’, ‘Proteomics’ (such as pSILAC), ‘Sequencing’ (i.e., RNA-Seq, HITS-CLIP, PAR-CLIP), ‘Degradome-Seq’, and ‘Other’ (e.g. ELISA, RACE, immunohistochemistry, etc.). For each entry, the experiment outcome (positive or negative), the type of association (direct or indirect), the type of regulation (up-regulation or down-regulation), the binding site, as well as the outcome for each specific methodology were inserted into the database by the curators. A small excerpt from the manuscript was also added, which was judged to include important information regarding the experiment.

Raw data sets from high-throughput experiments deposited in relevant repositories or provided in supplements of seven publications were analyzed (four microarray [24, 101, 55, 25], two HITS-CLIP [16, 108], one PAR-CLIP [28]). PAR-CLIP and HITS-CLIP data consist of genomic coordinates specifying potential positions of miRNA-binding sites. Each position is scanned for complementary sequence to the seed of all known miRNAs and if such occurrence is found, the gene is noted as targeted by the corresponding miRNA. For the PAR-CLIP data, the seed search is refined by limiting the scanned region within 10 nucleotides around the T to C mutation. The mutation has been shown [28] to occur near the crosslink site of the Ago protein to the mRNA and therefore indicates with higher accuracy the position of the miRNA-binding site.

The new database includes 65,814 experimentally validated miRNA-gene interactions, which are extracted from relevant literature by DIANA TarBase curators. This is a 50-fold increase of entries from the previous DIANA TarBase version and a 16.5- to 175-fold increase from all the other available manually curated databases.
DIANA TarBase v.6 accommodates a significant number of outcomes procured from state-of-the-art high-throughput studies. Importantly, DIANA TarBase hosts data derived from 3 CLIP-Seq and 12 Degradome-Seq studies, which is a major increase compared to the eight studies supporting StarBase, a database dedicated to collecting data from those methodologies.

The new TarBase interface attempts to balance ease of use and functionality. Users can browse through numerous miRNA targets and explore the outcomes of hundreds of experimental studies in a simple and intuitive way. However, simplicity should not prevent users from performing complex functions, like processing results using several types of filters or executing combined searches. Such features are usually absent in most previous miRNA target databases.

From the DIANA TarBase search box (Figure 4.9) users can search for targets by inserting one or more miRNA identifiers, gene identifiers or a combination of miRNAs and genes (combined search). For users’ convenience, there is an extensive support for numerous miRNA and gene identification nomenclatures such as miRNA name or MIMAT id, for miRNAs and gene name, Ensembl id, or RefSeq id for genes. In cases of spelling errors or any other ambiguity in the user’s input, the system presents automatic suggestions. In such cases, the user can select any of the options provided to efficiently complete the database query.

Figure 4.9: A screenshot of DIANA TarBase v.6 interface.

Following the submission of a user query, the system displays related targets and all available relevant information in the form of an enhanced expandable list below the DIANA TarBase search box. Each list row corresponds to an experimentally verified miRNA target. Each entry is accompanied with a significant amount of information related to the gene, the miRNA and their interaction. Supporting experimental data are also provided, such as the methods used for validation, the outcome of each method and the DIANA microT v.4 prediction score. The latter score is also hyperlinked to the corresponding DIANA microT page, to provide detailed information about the target prediction.

Each entry in the new result interface is expandable, rendering further information in an easy and intuitive way. The user can show or hide this information by simply clicking on the arrows next to each entry. The extra information presented includes (a) detailed description of the involved gene and miRNA (name, description,
relevant MeSH terms, transcripts, etc.) and (b) the list of publications that support
the interaction, along with details about the experimental methods utilized and their
outcomes. Gene and miRNA information is enhanced with relevant KEGG path-
ways, links to external sources of data such as UniProt and Ensembl, and coupled
with accession codes to relevant external databases such as RefSeq. Interaction data
are supported with notations for direct/indirect interaction, up/down-regulation of
the target, cell and tissue types used in the experiment and the binding region of the
miRNA to the target. Moreover, a PubMed hyperlink to the supporting publication
is provided along with an excerpt of the manuscript text denoting useful information
regarding each interaction.

The Advanced Search feature of DIANA TarBase interface assists users to cus-
tomize the result list according to their specific needs. Users can enable this feature
by clicking on the gear wheel button at the right of the search box (Figure 4.9)
to activate and display an extensive set of filtering options. Users can also cus-
tomize the results list by selecting the desired value for each option. Option se-
lection includes (a) species, (b) experimental validation method (i.e. qPCR, mi-
acroarray, etc.), (c) regulation type (i.e. up/down), (iv) type of interaction (i.e.
 direct/indirect), (v) validation outcome (i.e. positive/negative), (vi) publication
year, (vii) DIANA microT prediction score.

Furthermore, DIANA TarBase v.6 provides the option to the user to access tar-
gets indexed in all other manually curated databases. Specifically, DIANA TarBase
integrates entries from miRecords, miRTarBase, and miR2Disease (see also Chap-
ter 4.4.1). This functionality is inactive by default but the user can easily enable
the integration module from the advanced search menu. Following activation, the
records from the external databases are incorporated in the result list. Each entry
is accompanied with an explicit mention of database of origin. All statistics men-
tioned previously regarding the DIANA TarBase database did not involve integrated
targets or external sources of miRNA interactions.

Finally, the DIANA TarBase v.6 user has the ability to denote erroneous entries
with the click of a button or submit entries from an experiment by utilizing easy
to use active forms. Note that all DIANA TarBase v.6 user interfaces were imple-
mented in PHP using well established design patterns for Web development (like
MVC model, Active Records, etc). All the required data were stored in a relational
database (MySQL was used). The service ViMa provided by GRNET was used to
host the Web tool at http://diana.imis.athena-innovation.gr/DianaTools/
index.php?r=tarbase.

4.4.3 Automatic extraction of miRNA targets: towards DI-
ANA TarBase v.7

A major issue for all manually curated databases is that there is a large cost to keep
their data up-to-date. In the case of DIANA TarBase, the updates are very expensive
because human curators must continuously discover publications that may contain
information about the targets of miRNA molecules and, then, read each of these
publications to identify possible verified miRNA-gene interactions. However, usually,
the labs that manage curated databases with miRNA-gene interactions do not have
enough funding to hire experts dedicated to the task of continuous data curation.
Therefore, eventually, most of these databases are cursed to become outdated for
In order to help both DIANA TarBase and other similar databases to preserve their data up-to-date without requiring important additional funding, we developed a system that scans a given set of publications and discovers inside them sentences that may encode interaction between a miRNA and a gene. A prototype implementation of this system and a preliminary evaluation based on the interactions contained in DIANA TarBase v.6 was performed in [95]. Our plan is to develop a more stable version of this system, perform thorough experiments on its performance and, then, release it as an open-source project.

Figure 4.10 demonstrates the process followed by the system to identify real miRNA-gene interactions contained in a set of publications. The input is a set of publications in HTML/XML format. Such files can be downloaded from PMC database of NCBI. The system parses the publication files and extracts those sentences that contain at least a miRNA term and a gene term. The miRNA identification is done using a relatively simple grammar since there is official nomenclature that is used with some standard variations (for details see [95]). For the gene identification special dictionaries are used (e.g., Entrez Gene and HGNC for human genes, MGI for mouse genes, etc). Sentences that contain references to other publications are not considered since it is assumed that their content refers to the finding of the publication that is referenced. Moreover, sentences that contain pairs which do not appear together to any other sentence of the same publication are also not considered. This is because there are some random co-occurrences of miRNA and gene terms. We observed that if a study suggests that there is a target of a miRNA molecule in a particular gene, then, usually, the miRNA and the gene appear together many times in the text.

NLP is performed on the remaining sentences in order to extract information required to build feature vectors for each miRNA-gene pair that will be used later by a classifier which decides if a pair corresponds to a real miRNA-gene interaction or not. During NLP, each word of the sentence is stemmed (by using the Morpha Stem-
mer [64]) and its part of speech is recognised (by using the Stanford Tagger\(^7\) [94]). Furthermore, the dependencies between the words of each sentence are extracted by using the Dependency Parser\(^8\) [44] of Stanford NLP. Finally, each sentence is separated into its phrases by a custom phrase chunking software.

All the data extracted by NLP are used to form feature vectors for each miRNA-gene pair. These feature vectors capture common text patterns that appear in sentences which describe experimentally verified miRNA-gene interactions. Each feature vector extracted from each publication is given to a Binary Maximum Entropy classifier. The output of this classifier for the feature vector of a miRNA-gene pair contained in a given publication corresponds to the probability that the publication contains experimental results which verify that this miRNA targets this gene. If this probability is greater than 0.5 we assume that the reported interaction is real. Note that the classifier was trained using data from DIANA TarBase v.6. For more details, see [95].

Table 4.1 presents the performance of the above described system based on some preliminary results presented in [95]. During these experiments we observed that many of the miRNA-gene verified interactions are not reported in the text of the publication but in supplementary figures, Excel sheets, or tables. Considering these interactions that are not encoded in the text of publications was out of the scope of [95] and can be part of future work. In order to evaluate the performance of the implemented prototype system we had to exclude from our experiments most of the publications that contain interactions many in supplementary material. We assumed that it is unlikely that a publication reporting more than 10 interactions encodes this information in text. Thus, we measured precision, recall, and F1-score for our system using all publications in our dataset that contain at most 2, 3, 5, or 10 interactions. Note that our dataset contains 1,167 publications downloaded from NCBI PMC and their miRNA-gene interactions were collected from DIANA TarBase v.6. This dataset was divided into a training set (85% of the publications) and a test set (the rest 15%). The measurements in Table 4.1 refer to the performance of the implemented prototype system on the test set.

<table>
<thead>
<tr>
<th>Max. #inter. per publ.</th>
<th>Precision (%)</th>
<th>Recall (%)</th>
<th>F1-score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>76.9</td>
<td>69.1</td>
<td>72.6</td>
</tr>
<tr>
<td>3</td>
<td>79.9</td>
<td>66.2</td>
<td>72.3</td>
</tr>
<tr>
<td>5</td>
<td>76.7</td>
<td>60.5</td>
<td>67.6</td>
</tr>
<tr>
<td>10</td>
<td>71.8</td>
<td>55.9</td>
<td>62.8</td>
</tr>
</tbody>
</table>

Table 4.1: Preliminary evaluation of the automated extraction of miRNA-gene interactions.

It is evident that the prototype system is accurate in a satisfactory level to provide suggestions to the DIANA TarBase curators. By considering also interactions reported in supplementary material of the publications the system will be even more practical.

\(^7\)http://nlp.stanford.edu/software/tagger.shtml

\(^8\)http://nlp.stanford.edu/software/lex-parser.shtml
4.5 DIANA mirPath: Discovering the role of miRNAs in pathways

In biochemistry, metabolic pathways are series of chemical reactions occurring within a cell. In each pathway, a principal chemical is modified by a series of chemical reactions. Enzymes catalyze these reactions, and often require dietary minerals, vitamins, and other cofactors in order to function properly. While the participation of genes in pathways is relatively well-studied, the same does not hold for the miRNAs.

miTalos [47] is a software can be used for the analysis of a subset of human signaling pathways. It identifies targets using five different external miRNA target prediction algorithms while also considering expression data. Moreover, miRTar [30] can be utilized for investigating alternatively spliced miRNA targets, which are identified by integrating external prediction algorithms. miRTar can also perform gene set enrichment analysis for the identification of miRNA targeted pathways. GeneTrail [6] is a web server hosting gene set enrichment and over-representation capabilities against various databases, such as GO and KEGG. GeneTrail has been extended with a tool, which queries miRNA identifiers against the MicroCosm Targets database for putative gene targets [7]. It provides many options during the gene enrichment analysis process, such as P-value thresholds and multiple testing significance correction but is lacking miRNA-specific functionalities, such as information regarding binding positions and binding type. Finally, DIANA mirPath v.1 [73] was one of the first available applications focused on the enrichment analysis of predicted target genes, capable of detecting pathways targeted by single or multiple miRNAs.

DIANA mirPath v.2 Webserver [100] is an entirely redesigned Web server with many new features. It aims to significantly increase the accuracy of utilized algorithms and statistics, as well as to enhance computational speed, compared to the previous DIANA mirPath version. Importantly, DIANA mirPath v.2 offers for the first time a series of tools specifically focused on miRNA-targeted pathway analysis. The user of DIANA mirPath v.2 can utilize predicted or experimentally validated targets; combine results with merging and meta-analysis algorithms; perform hierarchical clustering of miRNAs and pathways based on their interaction levels; as well as elaborate sophisticated visualizations, such as dendrograms or miRNA/pathway interaction heat maps, from an intuitive and easy to use Web interface. The new server provides additional information regarding pathogenic SNPs in predicted miRNA target sites. Furthermore, the reverse analysis module annotates all predicted or experimentally validated miRNAs targeting a selected molecular pathway.

DIANA mirPath v.2 is based on a new relational schema, specifically designed to accommodate this as well as future updates. miRNA and pathway related information was obtained from miRBase 18 and Kyoto Encyclopedia of Genes and Genomes (KEGG) v58.1 [39], respectively.

The DIANA mirPath v.2 interface (Figure 4.11) has been designed to be highly adaptable to different use-case scenarios and to provide results in real time. In order to perform the analysis, the user can select one or more miRNAs and the source of gene targets for each miRNA. Optionally, a list of expressed genes can be also loaded. Subsequently, the server presents the significantly enriched pathways, the targeted genes in each pathway and the number of miRNAs with positively identified targets for each pathway in the form of an interactive table.

In the case of predicted miRNA-gene interactions, the server provides a link to
the relevant DIANA microT v.5 server (Chapter 4.2) entries. There, the user can further inspect the predicted miRNA-gene interaction. Such interactions include the binding region, position and type. If a miRNA-gene interaction is experimentally validated, the server provides a link to the specific section of the DIANA TarBase v.6 (Chapter 4.4) website. The relevant entry provides information regarding the implemented experimental method used for validation and the supporting literature.

DIANA mirPath v.2 offers enriched KEGG pathway visualizations, where the targeted genes are specifically marked for easier inspection (Figure 4.12).

The new reverse search module can be used to identify all miRNAs which are predicted or experimentally validated to target a specific KEGG pathway. The module takes as input a KEGG pathway name or identifier and the source of miRNA targets. It subsequently identifies all the miRNAs targeting the selected pathway. The new module can become a powerful asset to scientists studying specific pathways. It can help examining validated relationships between pathways and miRNAs expressed in the available literature (DIANA TarBase targets) or to study novel miRNA-pathway interactions (DIANA microT targets). If the analysis is performed in silico, the user can determine the desired levels for sensitivity and precision by applying a DIANA microT v.5 score threshold.

DIANA mirPath v.2 provides also advanced features, statistics and visualization aids, which significantly increase the depth of the analysis and maximize the user’s influence on results’ calculation and presentation. Moreover, it is capable to perform follow-up analyses, such as hierarchical clustering of targeted pathways and miRNAs. DIANA mirPath v.2 realizes clustering of the selected miRNAs based on their influence on molecular pathways. It provides clustering of pathways based on the subset of miRNAs that target each pathway and the significance level of the interaction. The server performs the hierarchical cluster analysis based on a complete
linkage clustering method, where squared Euclidean distances are calculated as distance measures. The Web server can utilize absolute P-values in all calculations (option: ‘Significance Clusters’) or binary values (0=not targeted, 1=targeted), if the option ‘Targeted Pathways Clusters’ is selected. By utilizing these options, the algorithm can cluster together microRNAs targeting similar lists of pathways, as well as pathways, which are targeted by similar lists of microRNAs (Targeted Pathways Clusters); or take also into account the significance levels of the interactions (Significance Clusters) during the clustering process.

These advanced features can help the user identify relations between miRNAs or pathways depending on the effect size of the miRNA-pathway interactions. The Web server provides visualizations of the hierarchical clustering in the form of miRNA and pathway dendrograms.

Finally, the new DIANA mirPath server enables also the user to create advanced visualizations such as miRNAs versus pathways heat maps (Figure 4.13). Heat maps are graphical representations of data where values in a matrix are represented as colours. These intuitive visualizations have been proven useful in numerous fields, since they enable the users identify patterns in the data, which were not easily discernible when examining the parameters individually. Furthermore, they enable the visualization of a very large number of variables, their in-between relationships and their levels of interaction. The Web server utilizes the hierarchical clustering results on both axes (pathways and miRNAs), in order to construct the heat map visualization. As in the case of cluster analysis, the Web server provides two options for heat map calculation: ‘Significance Heat Maps’ and ‘Targeted Pathways Heat Maps’. The former involves the use of absolute P-values in all calculations, while the latter substitutes all P-values lower than the user defined threshold with 0, and 1 otherwise. With the use of these advanced tools, the user can examine numerous miRNA-miRNA, miRNA-pathway and pathway-pathway relationships. Such rep-
resentations can help researchers discover patterns and relationships hidden in the data. All plots are rendered in high resolution.

Figure 4.13: An example heat map from the DIANA mirPath’s interface.

All DIANA mirPath v.2 user interfaces were implemented in PHP using well established design patterns for Web development (like MVC model, Active Records, etc). All the required data were stored in a relational database (MySQL was used). Scripts that perform the involved statistical analysis were developed in R. Finally, the service ViMa provided by GRNET was used to host the Web tool at http://diana.imis.athena-innovation.gr/DianaTools/index.php?r=mirpath.

4.6 DIANA mirPub: Searching for miRNA-related publications

Identifying, among the millions of publications available in MEDLINE, those that are relevant to a specific miRNA of interest based on keyword search faces major obstacles. The main problem is related to the miRNA nomenclature. It is often used inconsistently, finding, for example, in different articles different names for the same miRNA. To make it worse, in some cases the nomenclature evolves. For instance, a publication may refer to miRNA names that are obsolete nowadays.

We present here DIANA mirPub, a database with an intuitive interface which provides a powerful tool to effectively search for miRNA related publications. DIANA mirPub’s search engine takes into account not only miRNA name variations, but also changes occurring in miRNA names and sequences (based on all available miRBase versions). It exploits both text mining on MEDLINE and data from curated databases to discover miRNA-to-publication associations. Moreover, DIANA mirPub follows a crowdsourcing approach: users may upload their own data to assist the manual curation of miRNA related publications. Another key feature
of mirPub is an interactive visualization service that illustrates intuitively the evolution of miRNA data. Other features provided include, tag clouds summarizing the relevance of publications to particular diseases, cell types, or tissues, and access to DIANA TarBase data (see Chapter 4.4) in order to oversee genes related to miRNA publications.

4.6.1 DIANA mirPub description

In this chapter, we describe the design and development of the mirPub database and Web application, and we demonstrate mirPub’s functionality and services. In particular, in Chapter 4.6.1.1 we show how we discover miRNA-to-publication associations. In Chapter 4.6.1.2, we present methods to capture miRNA data evolution by processing multiple releases of miRBase. Finally, in Chapter 4.6.1.3 we describe mirPub’s interface and basic features.

4.6.1.1 miRNA-to-publication associations

MirPub provides miRNA-to-publication associations using the following three sources:

1. A manually curated database based on already discovered miRNA-to-publication associations from miRBase, DIANA TarBase and mir2disease.

2. MEDLINE/PubMed, where we discover associations using text mining techniques on the titles, the abstracts, and the full texts (where they are available) of MEDLINE/PubMed papers.

3. User-contributed data, since mirPub users are able to contribute to its content by reporting problematic records or by suggesting miRNA-to-publication associations that do not exist in the database yet.

Note that, a key contribution here is that, for the mining task, we consider miRNA nomenclature evolution and the variations in miRNA naming. Next, we present our mining method in more detail.

The main idea of our text mining method on MEDLINE/PubMed data is illustrated in Figure 4.14. In brief, we search into the titles, the abstracts, and the full texts (where they are available) of all papers in order to find appearances of keywords that describe miRNAs, since these appearances imply microRNA-to-publication associations. To support efficient searching, we organize publication data in convenient
formats and data structures. Therefore, in the first step, we adopt Lucene to build an inverted index on MEDLINE/Pubmed baseline files (publication indexer component). The index provides efficient access to abstracts and titles that contain a keyword, and also provides other useful metadata info (e.g., the publication title, related MeSH terms, etc).

After the index construction, the miRBase parser is executed to produce all the miRNA and family names recorded in miRBase. In particular, all the .dat files from versions 1.0 to 18 of miRBase are used to get the hairpin and mature miRNA names, and the .fam file of miRBase version 18 is used to get the miRNA family names. We refer to the union of the previous names as the set of official names. We have to expand this set in order to contain also their variants, because papers don’t always contain exact occurrences of the official names. This is the task of the name extractor component.

Name extractor uses a set of predefined rules to produce variants of the official names. A first class of variants contains names from which the species prefix is omitted (in fact, this is a very common name usage). Another class contains the variants produced by replacing some tokens by others. For instance, the token “mir” in miRNA keywords is frequently replaced by the tokens “mirna” or “microrna”; the token “let” is replaced by “mir-let”, “mirna-let”, or “microrna-let”. Note that similar rules have been also used in ([105]) to identify miRNA names. The output of name extractor is the union of all official names and all their produced variants (we refer to this union as the set of miRNA keywords).

The final step is performed by the association extractor component. Its aim is to use all the miRNA keywords to probe the Lucene index. The result is a list of <keyword, PubMed ID> tuples which actually determine the miRNA-to-publication associations. We store these tuples in mirPub’s relational database.

Our text mining method may fail to discover some miRNA-to-publication associations. For instance, we do not take into account papers that contain miRNA keywords neither in their title nor in their abstract. Text mining in the full text can return many of the missing papers, however, this type of processing can be applied only to open-access papers, which are just a portion of the existing literature. Using data from curated databases (like miRBase, TarBase, etc) can help in this direction. However, even this solution is not adequate since such databases do not usually provide data updates regularly. Even if they do, the time elapsed between two subsequent updates can be long. For this reason, in mirPub, we encourage users to report new miRNA-to-publication associations, as well as errors. A specific protocol will ensure that each new reported association will be examined by a curator prior to a final acceptance in the database.

4.6.1.2 Capturing miRNA data evolution

Since miRNA research is in flux, new publications amend the results of previous work, thus, the content of related databases, like miRBase, should be modified to comply with these changes. As many of these changes involve the name of miRNAs, keeping track of the naming history of a miRNA molecule is important for any researcher searching for publications related to this molecule. This is because knowing older names of the molecule or other molecules having the same name in the past can help the researcher to both expand her search in order to get more results, and remove non-related papers from the result set. In mirPub, we have processed all
available miRBase versions to extract useful miRNA data evolution information. We elaborate more on this in the following paragraphs.

In miRBase, each mature and each hairpin has a name and a sequence, which can be different from version to version, and an accession number, which is fixed and identifies it uniquely through different miRBase versions. All these data, together with metadata information, are stored in versioned files with the file extension .dat, called, from now on, the dat files of miRBase. Dat files follow a format similar to EMBL format, having one record for each distinct hairpin accession. Each record contains the name, the sequence, the produced mature sequences, the related publications, etc., of one particular hairpin. The mature data are stored as sub-entries of the hairpins that produce them.

By comparing the dat file of a version to the dat file of the next version, all the hairpin and mature changes from one version to the other can be produced. miRBase also provides a diff file for every version, which can be processed to extract the same information. However, available diff files exist only for miRBase versions newer than 3.1. Therefore, all miRNA evolution data in mirPub are produced by comparing dat files of miRBase versions to each other (from versions 1.0 to 18). We used the provided diff files only to validate the results of the aforementioned analysis for versions newer than 3.1. Note that, miRBase does not provide accession numbers for matures in versions older than 6.0. Therefore, we capture changes that involve mature data only for miRBase versions newer than 5.1.

After analyzing all miRBase files, from version 1.0 to 18, we identified the following change types:

- For mature miRNAs and hairpins:
  1. NEW: a novel miRNA has been inserted in the current version of miRBase
  2. NAME: a miRNA name has been modified
  3. SEQUENCE: a miRNA sequence has been modified
  4. NAME-SEQUENCE: a miRNA has both its name and its sequence been modified
5. DELETE: a miRNA has been removed from miRBase, and its accession become obsolete

• Only for hairpins:

1. FORWARD: a miRNA accession is replaced by another one

• For mature-hairpin related pairs:

1. ADD MATURE-HAIRPIN ASSOC: a mature was found to be produced by a particular hairpin

2. REMOVE MATURE-HAIRPIN ASSOC: a mature was found to not be produced by a particular hairpin

We parsed all miRBase files, extracted data related to changes that occur in miRNAs, and stored these data in a relational database.

4.6.1.3 Features and interface

mirPub services are provided through a free access Web interface. In the next chapters, we describe in detail issues related to the functionality of mirPub's interface, showing, for instance, how to search for publications and to refine searching by using filters, how to navigate in miRNA data evolution timeline, and how the users can contribute to the content of mirPub.

4.6.1.3.1 miRNA publication search

The user can search for publications related to particular miRNAs by inserting keywords describing these miRNAs in a searchbox of the mirPub interface (see also Figure 4.17). mirPub maps these user keywords to the stored miRNA keywords (see also Chapter 4.6.1.1). If an exact mapping for a user keyword is not possible, then mirPub identifies its most similar miRNA keywords, and recommends them to the user. To measure keyword similarity, we adopt a variation of edit distance that assumes zero cost for deletions. Moreover, if a user keyword is mapped to more than one miRNA keywords, then mirPub presents all the available options to the user and asks him to select which of them to include into his search query. In any other case, mirPub produces the list of publications that are found to be relevant to the mapped keywords (see Figure 4.17). This list is sorted based on the date of publication (most recent papers appear first). Each item in the list is an expandable rectangle displaying (in its compact form) the title and the date of publication, and, if it is applicable, two links, one to PubMed and another one to TarBase. When the user expands the rectangle, more information about the publication becomes available: the sources used to associate the publication to the user's query and the MeSH diseases, tissues, and cells that are related to the publication.

Note that, when the user submits a set of keywords through the searchbox, mirPub expands this set by adding other related miRNA keywords. In particular, for each identified miRNA, mirPub adds to the set of search keywords the relevant family name, related names based on history, and all its known specific name variants (based on the rules presented in Chapter 4.6.1.1). All miRNA keywords used to produce the results are displayed as hyperlinks in the “used keywords” box on the right of the result list (see Figure 4.17). By moving the mouse pointer over any
of these keywords, its relevant publications are highlighted. The user can choose to keep only the results related to some of these keywords simply by clicking once on the corresponding hyperlinks. The selected keywords become highlighted. By clicking again on a selected keyword, the selection disappears. Note that the used miRNA keywords are organized into four disjoint categories: (a) hairpins, (b) matures, (c) families, and (d) keywords (see the corresponding labels in Figure 4.17). The “hairpins” (resp. “matures”) category contains exact hairpin (resp. mature) miRNA names from any miRBase version. These names are followed by an information button which activates a pop-up displaying the history of the miRNA related to this keyword (see also Chapter 4.6.1.3.2). The “families” category contains family names from the last miRBase release. Finally, the “keywords” category contains variants of miRNA names.

Besides presenting the related MeSH diseases, tissues, and cells in each entry of the result list, mirPub aggregates this information for all entries and produces two useful tag clouds: one visualizing the related MeSH diseases to the publications in the result set, and another one visualizing the related MeSH tissues and cells. The font size of a MeSH term depends on the number of its relevant papers in the result set.

4.6.1.3.2 Visualizing miRNA data evolution mirPub users can explore changes in miRNA names, sequences, etc with a tool that visualizes the timeline of those changes. This tool is accessible through the information button that accompanies each reference of a miRNA name in mirPub’s user interface (e.g., there are four such buttons in the “used keywords” box of Figure 4.17). Clicking on this button results in rendering a pop-up that contains the history of all matures and hairpins that have been associated with this name for at least one miRBase version.

Figure 4.15 presents the timeline produced for “hsa-mir-29b-1” and Figure 4.16 the timeline produced for “hsa-mir-98”. The former name used to be associated with the hairpin MI0000105 for miRBase versions 2.0 to 2.2, and with the hairpin MI0000107 from version 3.0 until now. Therefore, one timeline for each of these hairpins is displayed (see Figure 4.15). These timelines show that “hsa-mir-29b-1”
was originally assigned to MI0000107 after a NAME change during the update of miRBase to version 2.0 (the previous name of this hairpin was “hsa-mir-102-1”). Then, in version 3.0, the same name was assigned to MI0000105, while MI0000107 was renamed into “hsa-mir-29b-2”. On the other hand, the name “hsa-mir-98” is associated with the hairpin MI0000100 for all versions newer than 1.2. The timeline of changes of this hairpin (illustrated in Figure 4.16) shows that the name “hsa-mir-90” was assigned to it after a NAME change. Then, in version 7.0, miRBase modified the sequence of this hairpin. Note that, orange color is used in each timeline to highlight the states of hairpins that involve the name used to produce the timeline (i.e., “hsa-mir-29b-1” in Figure 4.15 and “hsa-mir-98” in Figure 4.16). Moreover, note that, for presentation reasons, the last hairpin state belongs to the last miRBase version, always, thus, a dummy change pointing to it (having the label “No Change”) is required in most cases.

4.6.1.3.3 User contributed data As mentioned in Chapter 4.6.1.1, in order to increase its precision and as an additional option to remain up-to-date, mirPub provides to the user the opportunity to upload data. In particular, there are two available operations: (a) requests for new miRNA-to-publication associations can be uploaded to be inserted in mirPub’s database, and (b) an existing miRNA-to-publication association can be reported as invalid. The former is done by clicking on the “Submit new results” button located below the search box (see at the right of Figure 4.17), while the latter is done by clicking on the “thumbs down” button located in the expanded view of each result (see also Figure 4.17). Both actions result in rendering a new page containing a form that collects user data. The uploaded data become part of mirPub’s database only after moderation performed by an expert curator. This final step is required to keep a high level of data quality.

4.6.2 Evaluation

In this chapter, first we analyze the mirPub database and demonstrate interesting statistics about the miRNA related publications and the miRNA data evolution. Then, we evaluate the effectiveness of mirPub for retrieving miRNA literature.

4.6.2.1 miRNA publications and data evolution statistics

Currently, mirPub’s database contains more than 57,436 distinct miRNA_keyword-PubMed_id pairs involving more than 7,471 distinct publications. Table 4.2 compares mirPub with the most known databases containing publications relevant to miRNAs in terms of the number of papers found to be associated with at least one miRNA. It is clear that mirPub contains the largest number of miRNA related publications compared to other databases.

Table 4.3 summarizes some interesting statistics for the publications stored in the mirPub’s database. In particular, the maximum and average number of miRNA keywords, MeSH diseases, and MeSH tissues & cells, per paper are presented. One paper contains 2,510 distinct miRNA keywords (from a total of 43,908). This is due to the fact that this paper is a large study of miRNA expression [48] done by sequencing over 250 small RNA libraries from 26 different organ systems and cell types. Thus, it contains a large number of miRNA related keywords. Table 4.3 also
Table 4.2: Comparison of several miRNA publication databases.

<table>
<thead>
<tr>
<th>Database</th>
<th>#papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirPub</td>
<td>7,471</td>
</tr>
<tr>
<td>miRBase</td>
<td>407</td>
</tr>
<tr>
<td>TarBase</td>
<td>1,392</td>
</tr>
<tr>
<td>mir2disease</td>
<td>519</td>
</tr>
<tr>
<td>miRCancer</td>
<td>573</td>
</tr>
</tbody>
</table>

presents the maximum and average number of sources used to retrieve each paper. Note that, mirPub uses 5 data sources: MEDLINE/PubMed, miRBase, TarBase, mir2disease, and user contributed data.

Table 4.3: Some interesting statistics of mirPub’s publications.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Max.</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>miRNA kwds/paper</td>
<td>2,510</td>
<td>9.71</td>
</tr>
<tr>
<td>diseases/paper</td>
<td>8</td>
<td>1.86</td>
</tr>
<tr>
<td>tissues &amp; cells/paper</td>
<td>8</td>
<td>1.58</td>
</tr>
<tr>
<td>sources/paper</td>
<td>4</td>
<td>1.32</td>
</tr>
</tbody>
</table>

More statistics can be found in Table 4.4, where the major scientific journals and MeSH terms related to miRNA papers are presented.

Table 4.4: The major journals containing miRNA related publications and the major MeSH diseases related to miRNAs based on publications.

<table>
<thead>
<tr>
<th>Journal</th>
<th>#papers (perc.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 PloS one</td>
<td>380 (5.09%)</td>
</tr>
<tr>
<td>2 Proc. of the Nat. Acad. of Sciences of the USA</td>
<td>241 (3.23%)</td>
</tr>
<tr>
<td>3 Cancer research</td>
<td>210 (2.81%)</td>
</tr>
<tr>
<td>4 The Journal of biological chemistry</td>
<td>208 (2.78%)</td>
</tr>
<tr>
<td>5 Biochemical &amp; biophysical res. comm.</td>
<td>150 (2.01%)</td>
</tr>
<tr>
<td>6 Blood</td>
<td>132 (1.77%)</td>
</tr>
<tr>
<td>7 Nucl. Acids Research</td>
<td>122 (1.63%)</td>
</tr>
<tr>
<td>8 Oncogene</td>
<td>115 (1.54%)</td>
</tr>
<tr>
<td>9 Cell cycle (Georgetown, Tex.)</td>
<td>88 (1.18%)</td>
</tr>
<tr>
<td>10 Journal of virology</td>
<td>86 (1.15%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MeSH Disease</th>
<th>#papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Neoplasm Invasiveness</td>
<td>317</td>
</tr>
<tr>
<td>2 Breast Neoplasms</td>
<td>305</td>
</tr>
<tr>
<td>3 Neoplasms</td>
<td>239</td>
</tr>
<tr>
<td>4 Lung Neoplasms</td>
<td>237</td>
</tr>
<tr>
<td>5 Liver Neoplasms</td>
<td>218</td>
</tr>
<tr>
<td>6 Carcinoma, Hepatocellular</td>
<td>198</td>
</tr>
<tr>
<td>7 Disease Models, Animal</td>
<td>194</td>
</tr>
<tr>
<td>8 Cell Transform., Neoplastic</td>
<td>168</td>
</tr>
<tr>
<td>9 Neoplasm Metastasis</td>
<td>156</td>
</tr>
<tr>
<td>10 Carcinoma, Squamous Cell</td>
<td>150</td>
</tr>
</tbody>
</table>

Figures 4.18 and 4.19, depict the number of changes introduced in each miRBase version for hairpin and mature miRNAs. In particular, the former illustrates the number of NEW and DELETE changes, while the later the number of NAME, SEQUENCE, and NAME-SEQUENCE changes for each miRBase version. In the stacked histograms, the light grey bars represent hairpin changes, while the dark grey bars represent mature changes. The general trend is that every new miRBase version introduces more changes than the previous ones.
Finally, an interesting observation is that a miRNA change may often trigger another one. For instance, the insertion of a new hairpin in miRBase is usually followed by the insertion of at least one mature miRNA. In fact, our analysis shows that 86.29% of the NEW mature changes follow a NEW hairpin change.

4.6.2.2 Evaluating literature retrieval

MirPub contains a subset of PubMed publications. Despite this fact, it is more effective to use this subset for searching for publications related to particular miRNAs. The strength of mirPub lies in the large set of miRNA-to-publication associations it contains. The most of these associations cannot be retrieved by PubMed itself because it hasn’t access to all the curated data mirPub has, and it cannot exploit the variants (see Chapter 4.6.1.1) and the history (see Chapter 4.6.1.2) of the miRNA names.

To evaluate mirPub’s effectiveness against PubMed to retrieve miRNA literature
we examined the results retrieved given a query set of 50 randomly selected miRNA names. We configured the search engines of both mirPub and PubMed to retrieve, for the same time period, only results that contain exact occurrences of the search terms. In mirPub this can be easily done by selecting the appropriate keywords from the “Used keywords” filter. Note that for PubMed we additionally exploit the miRNA-to-paper associations contained in the NCBI gene database since these data are also accessible through the PubMed result page.

The results are presented in Table 4.5. Consider that, under our scenario, both mirPub and PubMed have 100% precision. This means that all the retrieved papers are related to user’s search. The actual size of recall cannot be measured, since the entire set of related publications is unknown. However, the fact that mirPub returns more results than PubMed allows us to conclude that mirPub has higher recall. In particular, based on the number of retrieved papers, mirPub is expected to have recall that is more than twofold PubMed’s recall.

Table 4.5: Evaluation of mirPub’s effectiveness in miRNA literature retrieval in comparison to PubMed.

<table>
<thead>
<tr>
<th></th>
<th>#papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mirPub</td>
<td>134</td>
</tr>
<tr>
<td>PubMed</td>
<td>54</td>
</tr>
</tbody>
</table>

We also evaluated how knowing miRNA data evolution improves retrieval effectiveness. In particular, we asked an expert to use mirPub in order to search for 25 miRNAs. For the needs of the experiment we selected miRNAs having significant history. First we asked him to configure mirPub’s filters in the way he believes that he will get the most relevant to his search results. Then, we asked him to examine the history of all the matures and hairpins in the “Used keywords” box and, then, reconfigure the filters. Finally, he should provide judgements for the relevance of the retrieved results. Table 4.6 summarizes the results of this experiment. It is evident that being informed about the history of miRNAs can be very useful in miRNA literature search. In particular, searching without knowing miRNA history failed to retrieve the 30.23% of the relevant papers found otherwise. Moreover, the evolution knowledge helps the user to fix errors that rise during his search (e.g., the use of old keywords that refer to modified sequences) preserving high levels of precision (approx. 90.2% for our experiment).

Table 4.6: Evaluating how knowing miRNA data evolution improves searching for publications.

<table>
<thead>
<tr>
<th></th>
<th>#papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieved w/o ev. knowl.</td>
<td>90</td>
</tr>
<tr>
<td>retrieved with ev. knowl.</td>
<td>143</td>
</tr>
<tr>
<td>relevant</td>
<td>129</td>
</tr>
</tbody>
</table>

4.7 Conclusions

We performed substantial work to provide valuable tools to facilitate scientists that work in the field of miRNA research. To this end, we collected data scattered to
many scientific databases and publications, combined them and processed them to extract knowledge about the role of miRNA molecules in many life mechanisms. The results are distributed to the research community through a multitude of powerful tools having intuitive Web interfaces. By using them, life scientists are capable both to browse the field knowledge recorded in our databases and to perform many types of analysis on the stored data.

In particular, DIANA microT provides to life scientists predictions for the genes that are targeted by all the known miRNAs, based on the very accurate DIANA microT method. DIANA miRGen informs its users about the genomic locations of all miRNA transcripts and their expression behaviour. DIANA TarBase provides the most complete set of experimentally verified miRNA targets. DIANA mirPath investigates the role of miRNAs in the known metabolic pathways. Finally, DIANA mirPub assists life scientists in miRNA-related literature search.

The impact of the aforementioned tools have been evaluated through their use during the previous years. About 500 distinct researchers use them daily, more than 100 of them being registered users that benefit from the personalised features provided by our tools.
Chapter 5

Sequence alignment of long DNA reads

DNA read alignment is the core analysis for many applications in life sciences (see also Chapter 2.2.3). It consists of aligning DNA reads on reference genome sequences. The reads are small sequences extracted from biological samples using specialized equipment called sequencing machines.

Although many index structures and algorithms to accelerate DNA read alignment have been proposed in the past, most of them were optimized for reads extracted by past generation sequencing machines and do not perform well for reads produced by recent machines. In particular, new generation sequencing machines produce longer DNA reads (consisting of more than 200 symbols) and their reading accuracy decreases, making current filtering approaches for sequence alignment insufficient.

In this chapter, we present Hitmap, an indexing approach supporting efficient alignment for large read lengths and edit distance thresholds. In Chapter 5.1 we explain some preliminary concepts, we provide the motivation, and we present the state-of-the-art DNA read alignment algorithms. In Chapter 5.2 we introduce the Hitmap index structure and in Chapter 5.3 we describe an algorithm that exploits this structure to efficiently find all the alignments of a given query sequence into a data sequence. In Chapter 5.4 we provide experiments that evaluate the performance of our approach in comparison to the state-of-the-art approaches. Finally, in Chapter 5.5 we conclude our contribution.

5.1 Background

5.1.1 Useful notations and definitions

Only for this chapter, and for presentation reasons, given a sequence $S$, we denote as $S^{x:y}$ the subsequence of $S$ that starts at the $x$-th symbol and ends at the $(y-1)$-th symbol (attention: $0 \leq x, y \leq |S|$). Furthermore, we refer to its $x$-th symbol as $S^x$. Finally, we follow the convention that a capital letter, e.g., $X$, denotes an object (e.g., a sequence), whereas a small letter, e.g., $x$, corresponds to an associated number of $X$ (usually the length).

For the rest of the text, we refer to a data sequence (e.g., a genome reference sequence) as $D$ and to a query sequence to be aligned into it (e.g., a DNA read)
Figure 5.1: A sample data and query sequence along with the corresponding partitions.

as Q. Figure 5.1 presents a toy example data and query sequence of length $d = 19$
and $q = 12$, respectively; note that in real-life scenarios $d$ is in the order of billion
symbols and $q$ in the order of hundreds of symbols. Some useful definitions follow.

**Alignment.** A query sequence $Q$ is aligned at position $x$ of $D$ if $Q$ and $D_{x:x+q+\psi}$
are within edit distance $\epsilon$, for some $\psi \in [-\epsilon, \epsilon]$. We refer to $\epsilon$ as the alignment threshold,
and to the subsequence $D_{x:x+q+\psi}$ as the alignment site.

For instance, in the example of Figure 5.1 assuming that $\epsilon = 2$, there is an
alignment of $Q$ in $D$ at position $x = 6$. The alignment involves replacing $Q^7$ by the
symbol $C$ and inserting the symbol $G$ in position 3 of $Q$.

**Fragment.** Given a query $Q$ and an integer $f$, where $0 < f \leq q$, $Q$ is divided into
$\phi = \lfloor q/f \rfloor$ non-overlapping subsequences of length $f$, called fragments. We use $F_k$
to denote the fragment $Q_{k\cdot f:(k+1)\cdot f}$, where $k \in [0, \phi)$. Note that some of the symbols
at the end of $Q$ may not belong to any fragment.

For example, in Figure 5.1 considering $f = 3$, $Q$ is divided into $\phi = 4$ non-overlapping fragments: $F_0 = \text{GAT}$, $F_1 = \text{TAC}$, $F_2 = \text{AGG}$, and $F_3 = \text{CGG}$.

**$f$-gram.** An $f$-gram is simply a sequence of $f$ symbols. Naturally, a query fragment
is an $f$-gram.

**Appearance.** A fragment $F_k$ (of a query sequence $Q$), where $k \in [0, \phi)$, appears
inside a subsequence $D_{y:y+f}$ of the data, if there is $x \in [y, y+f]$ such that $F_k$ and
$D_{x:x+f}$ are identical. We refer to the subsequence $D_{x:x+f}$ as an appearance of $F_k$
at $x$.

Consider the subsequence $D_{7:14}$ of the data sequence in Figure 5.1. There is an
appearance of $F_1$ in this subsequence, since $F_1 = D_{10:13} = \text{TAC}$.

5.1.2 The pigeonhole principle

The pigeonhole principle is a simple observation upon which many sequence filtering
techniques are based. It basically states that if pigeons are put in pigeonholes, and
there are more pigeons than pigeonholes, then there exists at least one pigeonhole
that contains more than one pigeons. Its application to the sequence alignment
problem results in the following theorem.

**Theorem 5.1.** Consider a query $Q$ and a data $D$. Let $\epsilon$ be the alignment threshold,
$f$ be the size of query fragments, and $\phi = \lfloor q/f \rfloor$ their number.
If $Q$ is aligned at position $x$ of $D$ then at least $\phi - \epsilon$ fragments of $Q$ appear inside the corresponding alignment site.

The next corollary is essentially contraposition of Theorem 5.1 and, during the sequence alignment of a given sequence $Q$ in a data sequence $D$ with threshold $\epsilon$, provides the means to filter out positions of $D$ that do not contain any alignment of $Q$.

**Corollary 5.1.** If less than $\phi - \epsilon$ fragments of $Q$ appear in a subsequence $D_{y}^{z}$ of the data, then, $Q$ cannot be aligned at any $x \in [y, z - q]$.

### 5.1.3 Our motivation

In Chapter 2.2.3 we discussed in detail the production of DNA reads by sequencing machines and the need to align these reads in reference genomes. Using the notation introduced in Chapter 5.1.1 we can define the problem as follows.

**Definition 5.1 (DNA read alignment).** Consider a reference genome $D$, a DNA read $Q$, and an alignment threshold $\epsilon$. The objective of DNA read alignment is to find the positions of all the alignments of $Q$ in $D$.

From now on, we will use the terms “reference genome” and “data” interchangeably. The same holds for the terms “DNA read” and “query”.

At its heart, the read alignment problem is similar to the common approximate sequence matching problem (see Chapter 2.1.1.2). Therefore known sequence matching techniques were the first to be adapted in order to be used for this problem. In the past, many approaches based on hash-tables of grams or on compressed suffix arrays have been proposed to solve the problem (see Chapter 2.2.3). All these methods were designed to perform well for the alignment of small DNA reads, since the first generation of sequencing machines were capable to produce reads of $30 - 50$ symbols. However, the latest technological advances result in machines that produce significantly longer DNA reads sacrificing part of the reading accuracy (see Chapter 2.2.3).

Under the above mentioned conditions, the most state-of-the-art approaches fail to achieve adequate performance. WHAM was an approach introduced to fill this gap [54]. Experiments comparing WHAM to other state-of-the-art approaches revealed that it outperforms its competitors both for reads of small length ($30 - 50$ symbols) and for reads of slightly larger length (100 symbols). Moreover, WHAM was found to perform well even for scenarios of aligning reads using relatively loose alignment thresholds.

However, WHAM suffers from two important drawbacks. The first is that its original version allows searching for alignments that satisfy either (a) a given Hamming distance threshold or (b) three given thresholds, each of which determines the maximum number of insertions, deletions, and mismatches, respectively. Yet, using an edit distance threshold is the most popular, intuitive, and powerful choice. WHAM’s second drawback is that although it performs better than the other state-of-the-art short read alignment approaches, its performance during aligning reads of length larger than 100 symbols is not adequate. This is important since some modern sequencing machines produce DNA reads of $> 400$ symbols (some others
provide reads of thousands of symbols) and this is a trend we expect to continue (see Chapter 2.2.3).

Our intention was to provide a new approach that can efficiently align reads produced by modern sequencing machines. Therefore, our approach must perform well for reads that contain > 100 symbols and must maintain its benefits even in cases where the reading accuracy of the sequencing machine significantly reduces. Moreover, for the convenience of life scientists, our method should support edit distance thresholds.

To evaluate the performance of this method we adapted WHAM to support edit distance in order to form its main competitor. As an extra competitor we used RBSA [75] a state-of-the-art method known to support edit distance and capable to align long reads. In the following chapters we describe in detail our proposed approach and we present experiments that evaluate its performance against the above mentioned competitors.

5.2 The Hitmap index

In Chapter 5.2.1, we first present some observations and basic concepts upon which Hitmap index relies, such as data parts, position covers, and cover classes. Then, in Chapter 5.2.2, we describe in detail the Hitmap index structure and its construction for a given data (reference) sequence. Finally, in Chapter 5.2.3, we present a compression method used to reduce the memory footprint of Hitmap.

5.2.1 Basic Concepts

Consider a data sequence $D$, a query $Q$, the alignment threshold being $\epsilon$, and the selected size of query fragments being $f$ (with $\phi$ being the number of these fragments).

**Parts.** The most important concept related to the Hitmap index is that of the part, which is simply a $D$ subsequence of length $\pi = f + 2 \cdot \epsilon$. A part is longer than a fragment by exactly two times the alignment threshold $\epsilon$. The reason is to accommodate for insertions and deletions so that there is always at least one part bracketing any query fragment appearance, as will later become clear.

The Hitmap index stores information regarding all possible parts of $D$ (overlaps are necessary and allowed). For easy reference to parts, we organize them in a conceptual two-dimensional structure, and thus use two coordinates to unambiguously refer to them. In particular, a part is denoted as $\Pi[i][j]$ and corresponds to the data subsequence $D^{i+j+F(i)+1}$ of length $\pi$, where coordinate $i$ ranges in $[0, \pi)$, and coordinate $j$ in $[0, \lceil d/\pi \rceil)$. Note that, to guarantee that the parts at coordinate $j = \lceil d/\pi \rceil - 1$ are defined, we extend $D$ by adding $(\pi - 1) + \lceil d/\pi \rceil \cdot \pi - d$ special symbols (not belonging to the alphabet) at the end.

We refer to the set of parts that have common $i$ coordinate (a row of parts) as the $i$-th partition, denoted as $\Pi[i]$. There exist $\pi$ partitions, exactly as many as the length of a part. The reason being that the parts in partition $\Pi[i+1]$ can be obtained from those in partition $\Pi[i]$ by sliding their positions by one symbol; since a part has $\pi$ symbols, there exist $\pi$ possible slides. Note that each partition contains only non-overlapping parts.
Considering again the example of Figure 5.1, for \( f = 3 \) and \( \epsilon = 2 \) (the value of \( \epsilon \) is intentionally very high for illustration purposes), we set the part length equal to \( \pi = 3 + 2 \cdot 2 = 7 \). Then, there exists \( \pi = 7 \) partitions and each partition contains \( [d/\pi] = 3 \) parts of length 7. Each part is represented by a dashed rectangle. The extra symbols added to the end of \( D \) are denoted as $s$'s.

**Covers.** The motivation for defining parts is to help us determine if the query can be aligned at a particular position of the data. To achieve this, we employ the pigeonhole principle and check if sufficient number of query fragments appear inside carefully selected parts. These parts belong to a set, termed cover, defined with respect to a position in the data as follows.

Given a position \( x \) in \( D \), we define the cover of \( x \), denoted by \( C^x = \{C_0^x, \ldots, C_{\phi-1}^x\} \), as an ordered set of \( \phi \) parts such that \( C_k^x = \Pi[i_k^x][j_k^x] \), where \( i_k^x = mod(x - \epsilon + k \cdot f, \pi) \), \( j_0^x = \left\lfloor \frac{x-\epsilon}{\pi} \right\rfloor \), \( i_k^x = mod(x - \epsilon + k \cdot f, \pi) \), and \( j_k^x = \left\lfloor \frac{x-\epsilon+k\cdot f}{\pi} \right\rfloor \), for \( k \in [0, \phi) \).

Note that the parts in a cover may belong to different partitions. For example, in Figure 5.1, the grey colored parts form the cover of position \( x = 6 \), i.e., \( C^6 = \{\Pi[4][0], \Pi[0][1], \Pi[3][1], \Pi[6][1]\} \).

Each part in the cover is selected to capture the possible appearance of a particular query fragment. More specifically, the \( k \)-th fragment \( F_k \) can be related to the \( k \)-th part \( C_k^x \) in a cover. Given this mapping, the following result holds due to the pigeonhole principle.

**Theorem 5.2.** If the number of fragments \( F_k \), for \( k \in [0, \phi) \), that appear in \( C_k^x \) is less than \( \phi - \epsilon \), then there is no alignment at position \( x \).

**Proof.** We will prove this theorem by contradiction. Suppose there is an alignment at \( x \). Then, based on Theorem 5.1, it holds that at least \( \phi - \epsilon \) fragments of \( \psi \) appear inside \( D^{x+\epsilon+\psi} \), where \( \psi \in [-\epsilon, \epsilon] \).

Let \( F_k \), the \( k \)-th fragment of \( \psi \), be one of the appearing fragments, where \( k \in [0, \phi) \). Since \( \psi \) is aligned at \( x \) with at most \( \epsilon \) errors, the beginning of an \( F_k \) appearance should be between \( x + k \cdot f - \epsilon \) (if \( \epsilon \) deletions are applied to \( \psi \) before \( F_k \) and \( x + k \cdot f + \epsilon \) (if \( \epsilon \) insertions are applied to \( \psi \) before \( F_k \)). Based on the previous, any appearance of \( F_k \) resides in \( D^{x+k\cdot f-\epsilon \cdot x+(k+1)\cdot f+\epsilon} \).

Consider now \( C_k^x \), the \( k \)-th part of \( x \)'s cover. By definition, \( C_k^x = \Pi[i_k^x][j_k^x] \), where \( i_k^x = mod(x - \epsilon + k \cdot f, \pi) \), \( j_k^x = \left\lfloor \frac{x-\epsilon+k\cdot f}{\pi} \right\rfloor \). Based on the modulo definition: \( mod(a, b) = a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b \).\( i_k^x + j_k^x \cdot \pi = x - \epsilon + k \cdot f + \left\lfloor \frac{x-\epsilon+k\cdot f}{\pi} \right\rfloor \cdot \pi = x - \epsilon + k \cdot f + \left\lfloor \frac{x-\epsilon+k\cdot f}{\pi} \right\rfloor \cdot \pi \cdot \pi = x - \epsilon + k \cdot f + f + 2 \cdot \epsilon = x + \epsilon + (k+1) \cdot f \).

Therefore, the data subsequence that corresponds to \( C_k^x \) is identical to the data subsequence into which resides any \( F_k \) appearance. Since we selected \( F_k \) randomly from the set of \( \psi \) fragments that appear inside the alignment at \( x \), this means that the previous observation holds for any of the, at least, \( \phi - \epsilon \) fragments that must appear inside the alignment at \( x \). Thus, there are at least \( \phi - \epsilon \) fragments \( F_k \) appearing in \( C_k^x \), for \( k \in [0, \phi) \) which results in contradiction.

Given a query \( \psi \), during searching for its alignments, we can use Theorem 5.2 to filter out areas of \( D \), for which it is guaranteed that they do not contain any alignment. In contraposition, we can find areas having the possibility to contain an alignment, based on the following corollary.

\[ \]
Corollary 5.2. If the number of fragments \( F_k \), for \( k \in [0, \phi) \), that appear in \( C_k^x \) is at least \( \phi - \epsilon \), then it is possible that there is an alignment at \( x \). We refer to \( x \) as a candidate alignment.

For instance, in the example of Figure 5.1, there is an alignment starting at position \( x = 6 \) and, indeed, there are \( 3 > (\phi - \epsilon) = 2 \) fragments that appear in \( C^6 \). Note that, although here we have an alignment, in other cases it could be just a false positive.

**Cover classes.** Covers belonging to various distinct positions exhibit a nice symmetry. Let \( X \) be the set of the \( d \) possible positions existing in the data sequence \( D \). Given an alignment threshold \( \epsilon \) and a part length \( \pi \), we define \( \pi \) distinct equivalence classes as follows: \( \alpha = \{ x \in X : \text{mod}(x - \epsilon, \pi) = \alpha \} \), \( \forall \alpha \in [0, \pi) \). Note that if \( x \) belongs to the equivalence class \( \alpha \), then it can be expressed as \( x = \alpha + \lambda \cdot \pi + \epsilon \), where \( \lambda \in [0, [d/\pi]) \).

We define the cover class \( \alpha \) as the set of covers for positions \( x \in [\alpha] \). In many cases, for simplicity, instead of saying that the cover of a position \( x \) belongs to a cover class \( \alpha \), we say that \( x \) belongs to \( \alpha \).

The next theorem states certain relationships that hold for the coordinates of the parts in covers \( C^{\alpha_1}, C^{\alpha_2} \):

\[
(1) \ i_{k}^{\alpha_1} = i_{k}^{\alpha_2} = i_{k}^{[\alpha]}
\]

\[
(2) \ j_{k}^{\alpha_1} - j_{0}^{\alpha_1} = j_{k}^{\alpha_2} - j_{0}^{\alpha_2} = \Delta j_{k}^{[\alpha]}
\]

Moreover, writing \( x_1 = \alpha + \lambda^{x_1} \cdot \pi + \epsilon \) and \( x_2 = \alpha + \lambda^{x_2} \cdot \pi + \epsilon \), where \( x_1 \leq x_2 \) and \( \lambda^{x_1}, \lambda^{x_2} \in [0, [d/\pi]) \), then, for any \( k \in (0, \phi) \), it holds:

\[
(3) \ j_{k}^{\alpha_2} = j_{k}^{\alpha_1} + \lambda^{x_2} - \lambda^{x_1}
\]

**Proof.** For the needs of this proof we consider two data positions \( x_1 = \alpha + \lambda^{x_1} \cdot \pi + \epsilon \) and \( x_2 = \alpha + \lambda^{x_2} \cdot \pi + \epsilon \), where \( x_1 \leq x_2 \) and \( \lambda^{x_1}, \lambda^{x_2} \in [0, [d/\pi]) \).

First we prove statement (1). It holds that \( x_2 = \alpha + \lambda^{x_2} \cdot \pi + \epsilon = \alpha + (\lambda^{x_2} - \lambda^{x_1}) \cdot \pi + \epsilon = x_1 + (\lambda^{x_2} - \lambda^{x_1}) \cdot \pi \). Therefore, \( \forall k \in (0, \phi) \) it holds that

\[
i_{k}^{\alpha_2} = \text{mod}(x_2 - \epsilon + k \cdot \pi, \pi) = \text{mod}(x_1 + (\lambda^{x_2} - \lambda^{x_1}) \cdot \pi - \epsilon + k \cdot \pi, \pi) = \text{mod}(x_1 - \epsilon + k \cdot \pi, \pi) = i_{k}^{\alpha_1}.\]

Then, we prove statement (3). For all \( k \in (0, \phi) \), it holds:

\[
j_{k}^{\alpha_2} = \left[ \frac{x_2 - \epsilon + k \cdot \pi}{\pi} \right] = \left[ \frac{x_1 + (\lambda^{x_2} - \lambda^{x_1}) \cdot \pi - \epsilon + k \cdot \pi}{\pi} \right] = \left[ \frac{x_1 - \epsilon + k \cdot \pi}{\pi} \right] + \frac{(\lambda^{x_2} - \lambda^{x_1}) \pi}{\pi} = j_{k}^{\alpha_1} + \lambda^{x_2} - \lambda^{x_1}.
\]

Finally, we prove statement (2). For all \( k \in (0, \phi) \), it holds:

\[
j_{k}^{\alpha_2} - j_{0}^{\alpha_2} = j_{k}^{\alpha_1} + \lambda^{x_2} - \lambda^{x_1} - j_{0}^{\alpha_1} - \lambda^{x_2} + \lambda^{x_1} = j_{k}^{\alpha_1} - j_{0}^{\alpha_1}.
\]

An immediate corollary of Theorem 5.3 is that two covers \( C^{\alpha_1}, C^{\alpha_2} \) within a class contain parts that are distributed in a similar manner over the set of partitions. In particular, (1) the \( k \)-th part in \( C^{\alpha_1} \) and \( C^{\alpha_2} \) belongs to the same partition, and (2) the difference in the \( j \) coordinate of two consecutive parts in covers \( C^{\alpha_1} \) and \( C^{\alpha_2} \) is equal. Therefore, we can imagine the cover of \( x_2 \) as the cover of \( x_1 \) shifted to the right by \( \lambda^{x_2} - \lambda^{x_1} \) parts, assuming \( \lambda^{x_1} \leq \lambda^{x_2} \).
As an example, consider positions $x_1 = 6$ and $x_2 = 13$ of the data sequence in Figure 5.1. Both belong to the equivalence class $[4]$, as $x_1 = 4 + 0 \cdot 7 + 2 = 6$ and $x_2 = 4 + 1 \cdot 7 + 2 = 13$. It holds that $C^{x_1} = \{\Pi[4][0], \Pi[0][1], \Pi[3][1], \Pi[6][1]\}$ and $C^{x_2} = \{\Pi[4][1], \Pi[0][2], \Pi[3][2], \Pi[6][2]\}$, i.e., $C^{x_2}$ is just $C^{x_1}$ shifted to the right by one part.

### 5.2.2 Index Structure

**Description of the structure.** There is a finite number of $|\Sigma|^f$ possible distinct $f$-grams that can be fragments of a query sequence. The *Hitmap* index structure $\mathcal{H}$ is a hash table encoding their appearances in the parts of the data sequence $D$.

In particular, $\mathcal{H}$ contains one entry for each $f$-gram that appears at least once in $D$. The key $\kappa$ of each entry is the $f$-gram itself and the value $\mathcal{H}(\kappa)$, or $\mathcal{H}_\kappa$ for simplicity, corresponds to an array of bitsets $\{\mathcal{H}_\kappa[0], \ldots, \mathcal{H}_\kappa[\pi - 1]\}$ encoding the appearances of $\kappa$ in the parts of $D$. More specifically, the bitset $\mathcal{H}_\kappa[i]$ corresponds to $\Pi[i]$, i.e., the $i$-th partition of $D$, and contains one bit for each of its parts. The value of $\mathcal{H}_\kappa[i][j]$ is 1 if $\kappa$ appears inside $\Pi[i][j]$, or 0 otherwise. We use the term *hit* to refer to any bit that its value is equal to 1.

Figure 5.2 illustrates some entries of the *Hitmap* index created on the data sequence of Figure 5.1. Since the sequence $\text{TAC}$ is contained by the parts $\Pi[0][1], \Pi[1][1], \Pi[2][1], \Pi[3][1], \Pi[6][0]$, it holds that $\mathcal{H}_{\text{TAC}}[0][1] = \mathcal{H}_{\text{TAC}}[1][1] = \mathcal{H}_{\text{TAC}}[2][1] = \mathcal{H}_{\text{TAC}}[3][1] = \mathcal{H}_{\text{TAC}}[6][0] = 1$. All the other bits of $\mathcal{H}_{\text{TAC}}$ are set to 0.

**Index construction.** Given the data sequence $D$, its *Hitmap* index $\mathcal{H}$ is constructed by sliding a window of size $f$ from the first to the last position of $D$, while recording the appearances of $f$-grams in the parts of $D$.

Let $W^x = D_{x:x+f}$ be the subsequence of $D$ bracketed by the window when its start is placed at position $x$ of $D$. An $f$-gram having the same sequence as $W^x$ will appear in all the parts that entirely contain $W^x$, thus the corresponding bits in $\mathcal{H}_{W^x}$ must be set to 1. Note that, for each position $x$, there are $\pi - f + 1 = 2 \cdot \epsilon + 1$ parts that contain entirely $W^x$.

For example, consider that the *Hitmap* $\mathcal{H}$ of the data sequence presented in Figure 5.1 is under construction and that the sliding window is placed at position
Then, $W^{10} = TAC$ and the parts $\Pi[0][1]$, $\Pi[1][1]$, $\Pi[2][1]$, $\Pi[3][1]$, and $\Pi[6][0]$ contain entirely $W^{10}$. Therefore, the bits $H_{TAC}[0][1]$, $H_{TAC}[1][1]$, $H_{TAC}[2][1]$, $H_{TAC}[3][1]$, and $H_{TAC}[6][0]$ are hits of $TAC$ and must be set to 1.

### 5.2.3 Index compression

Each Hitmap entry contains $\pi$ bitsets (one for each partition) each of which consists of $\lceil d/\pi \rceil$ bits (one for each part). As a result, an entry contains in total $d$ bits, which is a very large number for big data sequences. Fortunately, the bitsets in Hitmap are extremely sparse for the most real-life scenarios and, therefore, the index can be heavily compressed achieving very large compression ratios, without sacrificing query efficiency.

The main idea behind the compression scheme is to avoid storing large sets of consecutive zero bits. Moreover, there is another observation that can also help: Within the same Hitmap entry, the non-zero bits (hits) tend to appear at the same position across contained bitsets. Consider, for example, the array of bitsets of the entry $\mathcal{H}_{AGG}$ in Figure 5.2; all its hits appear at the 0-th bit of each bitset. The above mentioned pattern is very common since the same positions of different bitsets in the same Hitmap entry correspond to highly overlapping parts of the data sequence.

A side-effect of the previous observation is that, if we represent the array of bitsets of each entry as a matrix, where each bitset consists a row, then the most of the columns of this matrix are expected to be full of zeros. Because of this, we choose to store the bits of the Hitmap entries column-wise instead of row-wise and we avoid to store at all any columns that are full of zeros. Moreover, since, in most real scenarios, the columns are sparse themselves, we managed to reduce even more the required space by building a large pool of all the distinct existent bitsets and storing in the Hitmap entries only pointers to the records of the pool.

### 5.3 Querying the Hitmap Index

In this chapter, we describe an algorithm that utilises the Hitmap index to find all the alignments of a given query $Q$ into a given data $D$, where the alignment threshold is $\epsilon$.

#### 5.3.1 The Hitmap alrgorithm

Consider a query sequence $Q$ and an alignment threshold $\epsilon$. It is possible to filter out areas in $D$ that do not contain any $Q$ alignment by utilising $\mathcal{H}$, the Hitmap index of $D$. This filtering process consists of (a) probing $\mathcal{H}$ to retrieve the arrays of bitsets $\mathcal{H}_{F_0}, \ldots, \mathcal{H}_{F_{\phi-1}}$, where $F_0, \ldots, F_{\phi-1}$ are the fragments of $Q$ and (b) performing bitwise operations on the retrieved bitsets to examine the condition of Corollary 5.2 for all the positions of $D$. If the condition holds for a position $x$, then the area around of this position must be examined for possible alignments. Otherwise, $x$ is filtered out.

Consider the position $x$ of $D$ and its $k$-th cover part $C^x_k = \Pi[i_x^k][j_x^k]$. By definition (see Chapter 5.2.1) this part is responsible to capture the appearance of the $k$-th fragment of $Q$ (i.e., $F_k$) in case there is an alignment of $Q$ at $x$. The bit in the
Hitmap index that corresponds to this part is the bit $H_{F_k}[i_k^\beta][j_k^\beta]$. From now on, we refer to this bit as the $k$-th *cover bit* of $x$.

It is evident that the condition of Corollary 5.2 can be examined for $x$ by counting the number of its cover bits that are hits. If at least $\phi - \epsilon$ of them are hits, then there is a candidate alignment at $x$. For example, in Figure 5.1, position $x = 6$ is a candidate alignment since among its cover bits $H_{GAT}[4][0], H_{TAC}[0][1], H_{AGG}[3][1]$, and $H_{CGG}[6][1]$, three of them are hits (in particular, all except of the third, see Figure 5.2).

An interesting observation is the following: Let $x_1$ and $x_2$ be two positions of $D$ that belong to the same cover class $[\alpha]$. It follows from the first property of Theorem 5.3 that, for any $k \in [0, \phi)$, the $k$-th cover bit of both $x_1$ and $x_2$ belong to the same bitset $H_{F_k}[i_k^\beta]$. We refer to the bitsets $B^{[\alpha]}[0], \ldots, B^{[\alpha]}[\phi - 1]$, where $B^{[\alpha]}[k] = H_{F_k}[i_k^\beta]$ for any $k \in [0, \phi)$, as the *cover bitsets* of the class $[\alpha]$. The following theorem holds for the cover bitsets.

**Theorem 5.4.** Consider a position $x = \alpha + \lambda \cdot \pi + \epsilon$ that belongs to the cover class $[\alpha]$, and consider that we shift each of its cover bitsets $B^{[\alpha]}[k]$ by $\Delta j_k^{[\alpha]}$ positions to the least significant bit. Then, $x$ is a candidate alignment iff we can find at least $\phi - \epsilon$ set bits at the $\lambda$-th bit of the shifted cover bitsets.

**Proof.** Based on Corollary 5.2 there is a candidate alignment at $x$ iff $F_k$ appears in $C_k^\alpha$, for at least $\phi - \epsilon$ distinct values of $k \in [0, \phi)$. Since $H_{F_k}[i_k^\beta][j_k^\beta] = B^{[\alpha]}[k][j_k^\beta]$, i.e. the $k$-th cover bit of $x$, is the one that encode the possible appearances of $F_k$ in the part $C_k^\alpha$, this means that there is a candidate alignment of $x$ iff $B^{[\alpha]}[k][j_k^\beta] = 1$ for at least $\phi - \epsilon$ distinct values of $k$. If we shift each $B^{[\alpha]}[k]$ for $\Delta j_k^{[\alpha]}$ to the LSB, this means that $x$ is a candidate alignment iff $B^{[\alpha]}[k][j'] = 1$ for at least $\phi - \epsilon$ distinct values of $k$, where $j' = j_k^\beta = \Delta j_k^{[\alpha]} = j_0^\beta = \lfloor \frac{\epsilon}{\pi} \rfloor = \lfloor \frac{\phi + \lambda \cdot \pi}{\pi} \rfloor = \lambda$. Therefore, iff at least $\phi - \epsilon$ of the $\lambda$-th bit from the shifted cover bitsets are set, then $x$ is a candidate alignment.

And a direct result of the previous theorem is the following.

**Corollary 5.3.** Consider the cover class $[\alpha]$, and consider that we shift each of its cover bitsets $B^{[\alpha]}[k]$ by $\Delta j_k^{[\alpha]}$ positions to the least significant bit. Then, if we can find at least $\phi - \epsilon$ set bits at the $\lambda$-th bit of the shifted cover bitsets it follows that the position $x = \alpha + \lambda \cdot \pi + \epsilon$ is a candidate alignment.

Based on Corollary 5.3 we can find candidate alignments of a query $Q$ that are located at positions belonging to a particular cover class. Therefore, by applying Corollary 5.3 for all cover classes, it is possible to locate candidate alignments everywhere in $D$. The pseudocode in Figure 5.3 demonstrates the aforementioned approach. We refer to this algorithm as the *Hitmap algorithm*.

The input of the *Hitmap* algorithm is the *Hitmap* index $H$ of $D$, the query $Q$, the alignment threshold $\epsilon$, and the desired number of query fragments $\phi$ (the latter determines also the size of the fragments, $f$). The output of the algorithm is the set of all candidate alignment positions $A$.

The algorithm consists of a loop executed once for each cover class (lines 1-10). During each iteration, the algorithm, first, allocates space for an array of bitsets, dedicated to (eventually) keep the cover bitsets of the class (line 2). Then, the
hitmap()
Input: \( \mathcal{H}, Q, \epsilon, \phi \)
Output: \( \mathcal{A} \)
begin
01. foreach \( \alpha \) in \( [0, \pi) \)
02. \( B^{[\alpha]} \leftarrow \) new array
03. foreach \( k \) in \( [0, \phi) \)
04. \( F_k \leftarrow Q.\text{getFrag}(k, \phi) \)
05. \( B^{[\alpha]}[k] \leftarrow \mathcal{H}(F_k[i^{[\alpha]}_k], \Delta j^{[\alpha]}_k) \)
06. \( B^{[\alpha]}[k] \leftarrow \text{lsb\_shift}(B^{[\alpha]}[k], \Delta j^{[\alpha]}_k) \)
07. end
08. \( O_{\phi - \epsilon}^{[\alpha]} B^{[\alpha]} \leftarrow t.\text{occ\_bitset}(B^{[\alpha]}, \phi - \epsilon) \)
09. \( \mathcal{A}.\text{addCandidates}(O_{\phi - \epsilon}^{[\alpha]} B^{[\alpha]}, [\alpha]) \)
10. end
end.

Figure 5.3: The Hitmap algorithm.

algorithm extracts one-by-one all the \( Q \) fragments (line 4) and uses them to retrieve their cover bitsets for the current class \( [\alpha] \) (line 5). Subsequently, each cover bitset is properly shifted according to Corollary 5.3 (line 6). When all the cover bitsets are ready and stored in the array of bitsets \( B^{[\alpha]} \), the algorithm processes this array to find those cover bitset positions \( \lambda \in [0, \lceil d/\pi \rceil) \) for which it holds that \( B^{[\alpha]}[k][\lambda] = 1 \) for at least \( \phi - \epsilon \) distinct values of \( k \) (line 8). The latter is performed by the function \( t.\text{occ\_bitset}() \), which involves bitwise operations (described in Chapter 5.3.1.1) to produce a bitset \( O_{\phi - \epsilon}^{[\alpha]} B^{[\alpha]} \) for which \( O_{\phi - \epsilon}^{[\alpha]} B^{[\alpha]}[\lambda] = 1 \) iff \( \lambda \) is a bitset position for which the previously described condition holds. Finally, the bitset \( O_{\phi - \epsilon}^{[\alpha]} B^{[\alpha]} \) is processed to produce the candidate alignments of \( Q \) based on Corollary 5.3 and the candidates of all classes are added to the same set \( \mathcal{A} \).

In the next chapter we proceed into details on the bitwise operations used by the function \( t.\text{occ\_bitset}() \) to produce the bitset that encodes the candidate alignments of a class.

### 5.3.1.1 Efficient bitwise operations to find candidates

Before proceeding further, note that the symbols \( \oplus, \wedge, \text{ and } \vee \) are used to denote the bitwise-XOR, bitwise-AND, and bitwise-OR, respectively, while the symbols \( \oplus^b, \wedge^b, \text{ and } \vee^b \) are used to denote bit-XOR, bit-AND, and bit-OR.

Consider a set of bitsets \( B = \{B_0, \cdots, B_n\} \), where each \( B_i \) contains \( m \) bits, \( \forall i \in [0, n] \). We define the \( t \)-occurrences bitset of \( B \) as the bitset \( O_t^B \) for which it holds that:

\[
O_t^B[z] = \begin{cases} 
1 & \text{iff } B_i[z] = 1 \text{ for at least } t \text{ distinct values of } i \in [0, n] \\
0 & \text{otherwise}
\end{cases}
\]

Figure 5.4 illustrates a set of 5 bitsets (at the left) and their 3-occurrences bitset (at the right). Note that \( O_3^B[1] = 1 \) because there are 3 bitsets (\( B_0, B_1, \text{ and } B_3 \)) having set their bit at position 1. On the other hand, \( O_2^B[2] = 0 \) because there is only one bitset (\( B_1 \)) having set the bit at position 2.

Figure 5.5 shows an algorithm that calculates the \( t \)-occurrences bitset for a given set of bitsets. Its input consists of the set of bitsets \( B \) and the threshold \( t \). Its output is the \( t \)-occurrences bitset. Note that the function \( \text{bin}(\alpha, \beta) \) returns a bitset containing the binary representation of \( \alpha \) using \( \beta \) bits.

96
Figure 5.4: An example set of bitsets, its $t$-occurrences bitset, and the counter bitsets used for the $t$-occurrences bitset computation ($t = 3$).

After performing the essential initialisations (lines 1−3), we count the set bits for all possible bitset positions (lines 4−11), and, then based on the result, we calculate the $t$-occurrences bitset (lines 12−17). During counting, we make use of some auxiliary bitsets. First of all, we use $C = \{C_0, \cdots , C_{|C|}\}$, a set of $|C| = \lceil \log_2(n+1) \rceil$ bitsets, which are used to encode the number of bits that are set for any possible bitset position. We refer to these auxiliary bitsets as the counter bitsets.

The counter bitsets encode the numbers of bits that are set as follows. Consider the bits $C_0[z], \ldots , C_{|C|}[z]$, for some $z \in [0, m]$. If we assume that these bits form a “virtual bitset” having $C_0[z]$ as its least significant bit (LSB), $C_1[z]$ as its next significant bit, and so on, then, this virtual bitset contains the binary representation of the number of $z$-th bits that are set. We refer to this virtual bitset as the bit-counter of position $z$.

For example, consider $z = 4$ in Figure 5.4. It holds that there are 4 bits that are set at position 4, since $B_0[4] = B_2[4] = B_3[4] = B_4[4] = 1$. We can see that the bit-counter of position 4 is $C_2[4]C_1[4]C_0[4] = 100$, which is the binary representation of 4. We refer to $C_0$ as the LSB counter bitset (because it contains the least significant bits of all bit-counters) and to $C_{|C|}$ as the MSB counter bitset. E.g., in Figure 5.4, $C_0$ is the LSB counter bitset and $C_2$ is the MSB counter bitset.

Initially, the counter bitsets contain only zeros (line 2 in Figure 5.5). We update their bits one time for each of the bitsets $B_w$ in $B$, where $w \in [0, n]$ (lines 4−11). After updating the counter bitsets for $B_w$, they encode the number of set bits in each position for the bitsets $B_0, \ldots , B_w$, i.e., for the bitsets we have considered until that time. Consequently, after performing the update for $B_n$, the counter bitsets will encode the number of set bits for all the given bit positions.

Updating the counter bitsets (lines 4−11) is based on binary addition. In particular, to update the counter bitsets for $B_w$, we first binary-add each of the $B_w$ bits to its corresponding bit of the LSB counter bitset and, then, we propagate the remainders of all these additions to be binary-added to the bits of the next counter bitset. Then, the new remainders are propagated for further additions to the next counter bitset, and so on, until the bits of the MSB counter bitset are updated. Note that we use $R = \{R_0 \cdots R_{|R|}\}$, a set of $|R| = \lceil \log_2(n+1) \rceil$ auxiliary bitsets, in which we store the remainders of all the performed binary additions ($R_w$ contains the remainders of the binary addition performed on the counter bitset $C_w$). Furthermore, as the binary addition of two bits is their bit-XOR and the remainder of this addition is their bit-AND, we implement the needed additions using bitwise-
t_occ_bitset()
Input: $B, t$
Output: $O^t_B$
begin
#Initialisations
01. foreach $y$ in $[0, \lceil \log_2(n + 1) \rceil)$
02. \hspace{1em} $C_y \leftarrow \text{bin}(0, m + 1)$
03. end
#Count the set bits
04. foreach $w$ in $[0, n]$
05. \hspace{1em} $R_0 \leftarrow C_0 \land B_w$
06. \hspace{1em} $C_0 \leftarrow C_0 \oplus B_w$
07. foreach $y$ in $[1, \lceil \log_2(n + 1) \rceil)$
08. \hspace{1em} $R_y \leftarrow C_y \land R_{y-1}$
09. \hspace{1em} $C_y \leftarrow C_y \oplus R_{y-1}$
10. end
11. end
#Compute $t$-set flags
12. $t\text{bin} \leftarrow \text{bin}(t, \lceil \log_2(n + 1) \rceil)$
13. $O^t_B \leftarrow \neg \text{bin}(0, m + 1)$
14. foreach $y$ in $[0, \lceil \log_2(n + 1) \rceil)$
15. \hspace{1em} if $t\text{bin}[y] == 0$ then $O^t_B[y] \leftarrow O^t_B \lor C_y$
16. \hspace{1em} else $O^t_B[y] \leftarrow O^t_B \land C_y$
17. end
end.
Figure 5.5: Pseudocode of an algorithm to calculate the $t$-occurrences bitset of a given set of bitsets

XOR and bitwise-AND operations (lines 5 – 6 and 8 – 9).

For example, consider again the bitsets of Figure 5.4 and assume that we just updated the counter bitsets for $B_3$ and that we are going to do the same for $B_4$. In this step, the content of the bit-counter of position 4 is $C_2[4]C_1[4]C_0[4] = 011$, as three of the bitsets $B_0, B_1, B_2,$ and $B_3$ (which are considered until this step) have set bits in position 4. The next bitset $B_4$, has also set its bit at position 4. We add this bit to $C_0[4]$ (which is the corresponding bit of the LSB-bitset of the counter bitsets) and get $C_0[4] \leftarrow C_0[4] \oplus B_4[4] = 0$ and $R_0[4] \leftarrow C_0[4] \land B_4[4] = 1$. The produced remainder $R_0[4]$ is then added to the corresponding bit of the next significant counter bitset, hence we get: $C_1[4] \leftarrow C_1[4] \oplus R_0[4] = 0$ and $R_1[4] \leftarrow C_1[4] \land B_4[4] = 1$. In the final step, we get: $C_2[4] \leftarrow C_2[4] \oplus R_1[4]$, $R_2[4] \leftarrow C_2[4] \land R_1[4]$, and, thus, the bit-counter of position 4 is $C_2[4]C_1[4]C_0[4] = 100$. Recall that, in fact, we simultaneously perform the bit operations for all positions, by using bitwise operations (we refer to bit operations here only for comprehension reasons).

After having calculated the counter bitsets, we have to use them to compute the $t$-occurrence bitset. To do this, we have to compare the bit-counter of each position to $t$. If the bit-counter of position $z \in [0, m]$ is greater or equal to $t$, then we set $O^t_B[z] = 1$. Otherwise, we set $O^t_B[z] = 0$. In order to compute all the $t$-occurrences bits simultaneously, we convert $t$ into its binary representation $t\text{bin}$ (line 12 in Figure 5.5), we initialise $t$-occurrences bits to be full of 1s (line 13), and, then, we perform some bitwise operations between the counter bitsets and the $t$-occurrences bitset, based on the bits of $t\text{bin}$ (lines 14 – 17). Note that $t\text{bin}$ contains one bit for each counter bitset.

The main idea behind lines 14 – 17 is the following. We update $O^t_B$, one time
for each counter bitset, i.e., one time for each bit in the bit-counters. The \((y+1)\)-th update of \(O_t^B\), where \(y \in [0, \lceil \log_2(n+1) \rceil]\), is achieved by performing a bitwise operation \(\diamond\) between \(O_t^B\) and the counter bitset \(C_y\), i.e., \(O_t^B \leftarrow O_t^B \diamond C_y\). For each bit \(O_t^B[z]\) of the updated \(O_t^B\), where \(z \in [0, m]\), must hold that \(O_t^B[z] = 1\) iff \(C_0[z] \cdots C_y[z] \geq tbin[0] \cdots tbin[y]\), or \(O_t^B[z] = 0\) otherwise. Therefore, the bitwise operation \(\diamond\) must be selected to guarantee the previous condition. At the time before the \((y+1)\)-th update, \(O_t^B[z]\) contains bits determined by the first (i.e., least significant) \(y\) bits of \(tbin\) and the first \(y\) counter bitsets (or, equivalently, the first \(y\) bits of all the bit-counters). During this update we consider \(tbin[y]\) (i.e., the \((y+1)\)-th bit of \(tbin\)) and \(C_y\) (i.e., the \((y+1)\)-th counter bitset). Assume that \(tbin[y] = 0\). For each \(z \in [0, m]\), there are two cases: (a) if \(C_y[z] = 0 = tbin[y]\), thus, we do not modify \(O_t^B[z]\) (all the involved binary numbers remain the same because zeros were added after their previous most significant bit), and (b) if \(C_y[z] = 1 > tbin[y]\), then \(C_0[z] \cdots C_y[z]\) is greater than \(tbin[0] \cdots tbin[y]\). From the previous, derives that \(O_t^B[z] \diamond 0 = F[z]\) and \(O_t^B[z] \diamond 1 = 1\), thus \(\diamond\) must be the bit-OR and, consequently, \(\diamond\) must be the bitwise-OR. Following similar reasoning we can show that if \(tbin[y] = 1\), then \(\diamond\) must be the bitwise-AND.

### 5.3.2 Optimising bitwise operations

The bitwise operations involved in the execution of Hitmap algorithm are performed on large bitsets, therefore, they may be intensive. However, since the involved bitset are sparse (contain very limited number of set bits), there is room for some optimisations.

In particular, we can represent any bitset as a set of blocks of particular size. Then, the bitwise operations performed on two bitsets is translated into performing the same operations on their blocks. Note that, since the bitsets are sparse, the most of their blocks are expected to be empty, i.e., full of zeros. This is very useful because bitwise operations involving empty bitsets have expected results, therefore, these bitwise operations can be avoided.

Consider a non-empty block \(B\) and let \(B_\emptyset\) denote any empty block. Then, the following properties hold:

\[
B \land B_\emptyset = B_\emptyset \\
B \lor B_\emptyset = B \\
B \oplus B_\emptyset = B
\]

By using the above three properties, we avoid performing many bitwise operations resulting in improved performance for Hitmap. Note that the block size is a Hitmap parameter that can affect significantly the execution time. Finally, note that the way the bitsets are stored in the index (described in Chapter 5.2.3) is not modified; the compressed bitsets are converted into the block-based representation just before the performance of bitwise operations on them.

### 5.4 Evaluation

In this chapter we describe a set of experiments we performed to evaluate the performance of Hitmap index and algorithm in comparison to the state-of-the-art approaches under the scenario of long DNA read alignment. Our main competitors
are (a) the WHAM index [54], which is the stat-of-the-art for alignment of short DNA reads and (b) the RBSA index [75], which was found to behave well for large DNA reads in the past.

In Chapter 5.4.1 we discuss the experimental setting and the datasets used for the experiments, while in Chapter 5.4.2 we evaluate Hitmap’s performance against the aforementioned competitors.

5.4.1 Setup

System. We implemented all algorithms in C++, and ran the experiments on a dedicated Linux server having two Intel Xeon E5607 processors at 2.27GHz and 64GBs of main memory.

Datasets. For the comparative experiments of Chapter 5.4.2 that involve RBSA we used a small synthetic dataset containing 100,000 symbols, where each symbol was randomly selected based on the uniform Bernoulli model (i.e., each symbol has $1/|\Sigma|$ probability to occur and is selected independently of others). This decision was made since the construction of RBSA is computationally intensive (it involves calculating the edit distance between each reference sequence and each possible subsequence of the data). Even for the toy data sequence used for our experiments building the RBSA index required several hours for particular configurations. We refer to this dataset as SYNTH.

For the rest experiments we used data from the unmasked human genome downloaded from the Genome Reference Consortium (in particular, version GRCh38 was used). This dataset contains sequences of the 24 human chromosomes and it has total size of 3.14GB (these data consist of the sequences along with some metadata). Each chromosome sequence consists of 50−250 millions of symbols.

Note that in all cases (i.e., both synthetic and real datasets), the query sequences were randomly selected subsequences of the used dataset.

Index configurations. We performed substantial testing to ensure that each index is optimally configured for each experiment.

We found that, for the case of relatively large query sequences (> 50 symbols), WHAM performs better when it is configured to be identical with the classical Navarro gram index [68], i.e., when the number of the query fragments is selected to be equal to $\epsilon + 1$ (see also Chapter 2.1.1.3). This was not surprising since its creators follow the same approach in [54]. However, this is the first time that WHAM is adapted to work for edit distance thresholds (see Chapter 5.1.3), thus it was necessary to reveal the ideal configuration under this setting. Note that by increasing the number of fragments we observed worse performance in all cases, even in those when the achieved filtering was better. The reason is that in these cases the number of probes to the WHAM index increases dramatically.

For the RBSA index, we used a set of 500 randomly selected reference sequences. Each data position was related to 50 of the reference sequences based on a slightly modified version of the greedy approach described in [75]. Our modification was that, instead of making a random selection for the query sample, we used our query sequences for this job. Of course this modification is not fair for WHAM and Hitmap, since the constructed RBSA index achieves optimal filtering for the queries of our experiment using an a-priory knowledge about them. However, we used this non-realistic scenario just to demonstrate that, even under this optimal setting, RBSA
performs significantly worse than WHAM and Hitmap.

The Hitmap index was configured to have $\phi = \epsilon + 2$ or $\phi = \epsilon + 1$ in cases that the value $\epsilon/q$ was small. For the rest cases, more fragments should be used in order to achieve improved filtering. The bitset block size was set to 500 in all cases.

5.4.2 Comparison to WHAM and RBSA

In Chapter 5.4.2.1 we compare the performance of Hitmap to this of WHAM and RBSA using our synthetic dataset. Since Hitmap and WHAM are found to significantly outperform RBSA, we focus on their comparative evaluation in Chapter 5.4.2.2.

5.4.2.1 Three algorithms comparison

Figure 5.6 illustrates the performance of Hitmap approach against WHAM and RBSA under the case of long queries for the SYNTH dataset. Each subfigure corresponds to a different query length $q$. We performed experiments for $q = 500$ and 1,000. For each value of $q$ we measured the execution time (y-axis, presented in log scale) of the three algorithms for many different alignment thresholds $\epsilon$ that vary from $0.02 \cdot q$ to $0.12 \cdot q$ (x-axis). We do not examine alignment thresholds that are greater than 12% of the query length because they are not practical.

![Figure 5.6: Exec. time for Hitmap, WHAM, and RBSA for various alignment thresholds on SYNTH dataset, in case of large reads.](image)

It is evident that Hitmap outperforms both RBSA and WHAM for any of the examined alignment thresholds. In particular, in all cases, Hitmap is faster than RBSA for more than one orders of magnitude. Also Hitmap outperforms WHAM for one order of magnitude or more in all cases where $\epsilon = 0.10 \cdot q$. Note that for $\epsilon \geq 0.10 \cdot q$ and $q = 500$ and for $\epsilon \geq 0.08 \cdot q$ and $q = 1,000$, RBSA fails to filter out any part of the data, thus, the whole data sequence must be examined for possible alignments of $Q$.

While Hitmap is optimised for long reads, it achieves reasonable performance for short reads. In Figure 5.7 we present the execution time of Hitmap, WHAM, and RBSA for queries of lengths $q = 50$ and 100. The x-axis, again, corresponds to different alignment thresholds. Note that y-axis is presented in log scale.

Hitmap clearly outperforms RBSA for more than an order of magnitude. Moreover, although it performs worse than WHAM for the most cases of small reads, its
performance is always comparable to this of WHAM. Recall that the small read alignment is a well-studied topic and existing approaches like WHAM and [68] achieve almost optimal performance. This is because under such scenarios even loose filtering conditions (like those incorporated by WHAM and [68]) achieve skipping more than 99% of the data. On the other hand, Hitmap focuses on providing almost perfect filtering even in the difficult cases, where WHAM and RBSA filtering fail to exclude large parts of the data.

In Table 5.1 we present the $\phi$ value used for Hitmap in each of the above discussed experiments.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\epsilon/q$</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.02</td>
<td>$\epsilon + 1$</td>
</tr>
<tr>
<td>50</td>
<td>0.04</td>
<td>$\epsilon + 1$</td>
</tr>
<tr>
<td>50</td>
<td>0.06</td>
<td>$\epsilon + 1$</td>
</tr>
<tr>
<td>50</td>
<td>0.08</td>
<td>$\epsilon + 1$</td>
</tr>
<tr>
<td>50</td>
<td>0.10</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>50</td>
<td>0.12</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>100</td>
<td>0.02</td>
<td>$\epsilon + 1$</td>
</tr>
<tr>
<td>100</td>
<td>0.04</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>100</td>
<td>0.06</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>100</td>
<td>0.08</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>100</td>
<td>0.10</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>100</td>
<td>0.12</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.02</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.04</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.06</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.08</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.10</td>
<td>$\epsilon + 2$</td>
</tr>
<tr>
<td>500</td>
<td>0.12</td>
<td>$\epsilon + 2$</td>
</tr>
</tbody>
</table>

To sum up, Hitmap have been proven to perform better than WHAM and RBSA for long queries and consists the only adequate filtering solution as the contained to the query errors increase. Moreover, RBSA is clearly outperformed by the two other methods, thus, in the next chapter we are going to perform experiments on real datasets comparing only Hitmap and WHAM.

### 5.4.2.2 Hitmap vs WHAM for long queries

In this chapter, we compare the performance of Hitmap against this of WHAM for long queries on a real dataset. In particular, we conduct our experiments using as data the sequence of chromosome 19 of Homo sapiens (58,617,616 symbols long). Figure 5.8 shows the execution times of Hitmap and WHAM (the y-axis is in log scale, while the x-axis corresponds to different selection of alignment thresholds).

For the case of $q = 500$, Hitmap outperforms WHAM for approximately one order of magnitude for almost all cases. Exception consists the case $\epsilon = 0.02 \cdot q$,
where WHAM performs slightly better. The reason is that in this case, more time is consumed in index probing than in the verifications of the candidate alignments. Both methods achieve satisfactory filtering.

For the case of $q = 1,000$, Hitmap’s performance is about two orders of magnitude better than this of WHAM, for almost all cases. The only exception is, again, the case $\epsilon = 0.02 \cdot q$, for which Hitmap outperforms WHAM for one order of magnitude.

In Table 5.2 we present the $\phi$ value used for Hitmap in each of the above discussed experiments.

Table 5.2: Hitmap configuration for the experiments on chromosome 19 of Homo sapiens.

<table>
<thead>
<tr>
<th>$q$</th>
<th>$\epsilon/q$</th>
<th>$\phi$</th>
<th>$q$</th>
<th>$\epsilon/q$</th>
<th>$\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.02</td>
<td>$\epsilon + 2$</td>
<td>1,000</td>
<td>0.02</td>
<td>$\epsilon + 3$</td>
</tr>
<tr>
<td>500</td>
<td>0.04</td>
<td>$\epsilon + 3$</td>
<td>1,000</td>
<td>0.04</td>
<td>$\epsilon + 3$</td>
</tr>
<tr>
<td>500</td>
<td>0.06</td>
<td>$\epsilon + 4$</td>
<td>1,000</td>
<td>0.06</td>
<td>$\epsilon + 6$</td>
</tr>
<tr>
<td>500</td>
<td>0.08</td>
<td>$\epsilon + 5$</td>
<td>1,000</td>
<td>0.08</td>
<td>$\epsilon + 10$</td>
</tr>
<tr>
<td>500</td>
<td>0.10</td>
<td>$\epsilon + 10$</td>
<td>1,000</td>
<td>0.10</td>
<td>$\epsilon + 11$</td>
</tr>
<tr>
<td>500</td>
<td>0.12</td>
<td>$\epsilon + 10$</td>
<td>1,000</td>
<td>0.12</td>
<td>$\epsilon + 13$</td>
</tr>
</tbody>
</table>

To sum up, Hitmap have been proven to perform better than WHAM for the most cases for long queries and consists the only adequate filtering solution as the contained to the query errors increase.

## 5.5 Conclusions

We introduced Hitmap an index structure along with an algorithm that supports efficient alignment for large read lengths and error thresholds. Hitmap fills the gap in read alignment approaches since it outperforms the state-of-the-art in the case of long DNA reads while its performance for short read alignment remains similar to the best of short DNA alignment algorithms.
Chapter 6

Conclusions and Future Work

This thesis presented various methods for managing vast amounts of data from life sciences. Our focus was on two interesting problems: (a) the efficient miRNA target prediction and (b) the efficient alignment of long DNA reads. For the former problem, we managed to build methods and systems that provide miRNA targets in near real time. This was achieved through the combined use of novel approximate sequence matching methods and the adoption of Cloud-based approaches. For the latter problem, we introduced Hitmap a new index structure that make possible the efficient alignment of DNA reads in reference genomes, for any size of reads, even reads consisting of hundreds or thousands of symbols. Finally, we implemented a large variety of Web tools to facilitate scientists in the area of miRNA research. Our tools make available and easy-to-search to life scientists information that was previously scattered or did not exist at all.

In the remainder of this chapter, we discuss in more detail our contributions and we identify interesting aspects that we propose for future work.

6.1 Summary

Initially, we considered the problem of miRNA target prediction. Our objective was to provide to life sciences accurate prediction of miRNA targets in near-real time focusing on the state-of-the-art DIANA microT method. In this direction, we studied the sequence matching process that consists the first step of the method. We found that it incorporates a novel sequence matching query. We formalised this query introducing the ARSM problem. The objective of this problem is to retrieve all regional occurrences of a pattern in a data sequence. The matching regions of the pattern must contain a predetermined area of the pattern, the core. Moreover, the allowable deviation from the data sequence is stricter for smaller and looser for larger regions. To deal with the previous problem, we proposed PS-ARSM method. Our method takes advantage of the prefix and suffix overlaps of regions avoiding redundant computations. A detailed experimental evaluation showed that PS-ARSM is up to two orders of magnitude faster than existing techniques adapted to the ARSM problem.

However, accelerating the sequence matching step of target prediction methods is not enough to achieve near-real time performance. This is because these methods also incorporate some other computational intensive processes. To boost their execution, we followed the approach to distribute them in the nodes of a Cloud...
infrastructure. Hence, we designed two Cloud-based target prediction systems, Tar-Cloud and MR-microT. The former was developed using the framework of Microsoft Azure, while the later was a MapReduce implementation using the Hadoop framework. Based on our measurements both systems accelerate the prediction process, however MR-microT is superior because (a) it is platform-independent (it can be installed in any cluster of contemporary computational nodes), (b) it provides improved parallelisation of the involved tasks, and (c) it is designed to support without problems increased number of prediction requests.

Then, we performed substantial work to provide valuable tools to facilitate scientists that work in the field of miRNA research. To this end, we collected data scattered to many scientific publications and databases, combined them and processed them to extract knowledge about the role of miRNA molecules in many life mechanisms. The results are distributed to the research community through a multitude of powerful tools having intuitive Web interfaces. By using them, life scientists are capable both to browse the field knowledge recorded in our databases and to perform many types of analysis on the stored data. In particular, we developed (a) DIANA microT, which provides to life scientists predictions for the genes that are targeted by all the known miRNAs, (b) DIANA miRGen, that informs its users about the genomic locations of all miRNA transcripts and their expression behaviour, (c) DIANA TarBase, that provides experimentally verified miRNA targets, (d) DIANA mirPath, which investigates the role of miRNAs in the known metabolic pathways, and (e) DIANA mirPub, a tool assisting life scientists in miRNA-related literature search. During the development of DIANA TarBase we recognised the difficulties the curators face when they need to identify miRNA-gene interactions recorded in the text of relevant publications. This was the motivation to investigate the opportunities in automatic recognition of miRNA-gene interactions. The results of the preliminary evaluation make us hopeful about providing adequate suggestions to DIANA TarBase curators. Note that the impact of the aforementioned tools have been evaluated through their use during the previous years. About 500 distinct researchers use them daily, more than 100 of them being registered users that benefit from the personalised features we provide to them.

Finally, we introduced Hitmap an indexing approach that supports efficient alignment for long DNA reads lengths and relatively large error thresholds. Hitmap fills the gap in read alignment approaches since it outperforms the state-of-the-art in the case of long DNA reads while its performance for short read alignment remains similar to the best of short DNA alignment algorithms.

6.2 Future Work

During the course of this dissertation, we have identified the following interesting aspects that we propose as future work.

- Regarding the long DNA reads alignment, we believe that it is easy to adapt the Hitmap index so that it can be used by Cloud-based systems. A such development will make it possible to provide Web tools for DNA read alignment in the human genome in real time. Many labs that have alignment needs but do not have the funding to buy powerful computers could benefit from these tools. Moreover, since DNA reads come in batches, it would be interesting
to preprocess them in order to exploit any overlaps they have during their alignment in the reference genome.

- Another direction related to sequence matching on data from life sciences is the efficient storage of multiple genome sequences in a manner that exploits their overlaps and facilitates fast aligning of DNA reads simultaneously on all these genome sequences. There are many scenarios (e.g., analysis of species evolution, personalised medicine, etc) where the DNA reads need to be aligned not to one genome but in a set of genomes. These genomes could be either reference genomes of different species or the genomes of individuals from the same species. Storing each genome separately results in inefficient disk use since these sequences contain many overlaps. Thus, efficient storage techniques, that exploit the previous overlaps must be used. Although there exist some genome compression techniques most of them focus on optimising storage usage. However, another important issue is to index the stored genomes in order to efficiently provide the alignments of a given read inside all these genomes. Investigating this problem might be quite interesting.

- During the development of DIANA mirPub we found that the existing methods for ranking scientific publications suffer from many issues. For instance, many of them favor old publications since there are many publications citing them. However, this is not fair for novel publications since those of them that are important are going to attract most of their citations in the future. Predicting the significance of new publications is an interesting open problem that could help in this direction.

- The work performed in the field of automatic knowledge extraction from miRNA publications (Chapter 4.4.3) should be extended and evaluated by using more judgements from field experts on the suggested miRNA-gene interactions provided by our approach. Moreover, an approach to exploit interactions recorded in supplementary tables and figures should be provided.

In conclusion, we believe that there is a plethora of interesting and novel topics relevant to the efficient management of life sciences data. We hope that this thesis will be an instigation for further research in this area.
Bibliography


113

miRSel: Automated extraction of associations between microRNAs and genes from the biomedical literature.


115


