Journal Subline
LNBI 3680

Transactions on Computational Systems Biology II

Corrado Priami
Editor-in-Chief

Springer

Lecture Notes in Bioinformatics 3680

Edited by S. Istrail, P. Pevzner, and M. Waterman

Subseries of Lecture Notes in Computer Science

Corrado Priami Alexander Zelikovsky (Eds.)

Transactions on Computational Systems Biology II

Springer

Volume Editor

Alexander Zelikovsky
Georgia State University
Computer Science Department
33 Gilmer Street, Atlanta, GA, USA
E-mail: alexz@cs.gsu.edu

Library of Congress Control Number: 2005933892

CR Subject Classification (1998): J.3, H.2.8, F.1

ISSN 0302-9743
ISBN-10 3-540-29401-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-29401-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11567752 06/3142 5 4 3 2 1 0

Preface

It gives me great pleasure to present the Special Issue of LNCS Transactions on Computational Systems Biology devoted to considerably extended versions of selected papers presented at the International Workshop on Bioinformatics Research and Applications (IWBRA 2005). The IWBRA workshop was a part of the International Conference on Computational Science (ICCS 2005) which took place in Emory University, Atlanta, Georgia, USA, May 22–24, 2005. See http://www.cs.gsu.edu/pan/iwbra.htm for more details.

The 10 papers selected for the special issue cover a wide range of bioinformatics research. The first papers are devoted to problems in RNA structure prediction: Blin et al. contribute to the arc-preserving subsequence problem and Liu et al. develop an efficient search of pseudoknots. The coding schemes and structural alphabets for protein structure prediction are discussed in the contributions of Lei and Dai, and Zheng and Liu, respectively. Song et al. propose a novel technique for efficient extraction of biomedical information. Nakhleh and Wang discuss introducing hybrid speciation and horizontal gene transfer in phylogenetic networks. Practical algorithms minimizing recombinations in pedigree phasing are proposed by Zhang et al. Kolli et al. propose a new parallel implementation in OpenMP for finding the edit distance between two signed gene permutations. The issue is concluded with two papers devoted to bioinformatics problems that arise in DNA microarrays: improved tag set design for universal tag arrays is suggested by Mandoiu et al. and a new method of gene selection is discussed by Xu and Zhang.

I am deeply thankful to the organizer and co-chair of IWBRA 2005 Prof. Yi Pan (Georgia State University). We were fortunate to have on the Program Committee the following distinguished group of researchers:

Piotr Berman, Penn State University, USA
Paola Bonizzoni, Università degli Studi di Milano-Bicocca, Italy
Liming Cai, University of Georgia, USA
Jake Yue Chen, Indiana University & Purdue University, USA
Bhaskar Dasgupta, University of Illinois at Chicago, USA
Juntao Guo, University of Georgia, USA
Tony Hu, Drexel University, USA
Bin Ma, University of West Ontario, Canada
Ion Mandoiu, University of Connecticut, USA
Kayvan Najarian, University of North Carolina at Charlotte, USA
Giri Narasimhan, Florida International University, USA
Jun Ni, University of Iowa, USA
Mathew Palakal, Indiana University & Purdue University, USA
Pavel Pevzner, University of California at San Diego, USA

Gwenn Volkert, Kent State University, USA
Kaizhong Zhang, University of West Ontario, Canada
Wei-Mou Zheng, Chinese Academy of Sciences, China

June 2005 Alexander Zelikovsky

Table of Contents

LNCS Transactions on Computational Systems Biology Editorial Board

What Makes the Arc-Preserving Subsequence Problem Hard?*

Guillaume Blin[1], Guillaume Fertin[1], Romeo Rizzi[2], and Stéphane Vialette[3]

[1] LINA - FRE CNRS 2729 Université de Nantes,
2 rue de la Houssinière BP 92208 44322 Nantes Cedex 3 - France
{blin, fertin}@univ-nantes.fr
[2] Universit degli Studi di Trento Facolt di Scienze - Dipartimento di Informatica e Telecomunicazioni Via Sommarive, 14 - I38050 Povo - Trento (TN) - Italy
Romeo.Rizzi@unitn.it
[3] LRI - UMR CNRS 8623 Faculté des Sciences d'Orsay, Université Paris-Sud
Bât 490, 91405 Orsay Cedex - France
vialette@lri.fr

Abstract. In molecular biology, RNA structure comparison and motif search are of great interest for solving major problems such as phylogeny reconstruction, prediction of molecule folding and identification of common functions. RNA structures can be represented by arc-annotated sequences (primary sequence along with arc annotations), and this paper mainly focuses on the so-called *arc-preserving subsequence* (APS) problem where, given two arc-annotated sequences (S, P) and (T, Q), we are asking whether (T, Q) can be obtained from (S, P) by deleting some of its bases (together with their incident arcs, if any). In previous studies, this problem has been naturally divided into subproblems reflecting the intrinsic complexity of the arc structures. We show that APS(Crossing, Plain) is **NP**-complete, thereby answering an open problem posed in [11]. Furthermore, to get more insight into where the actual border between the polynomial and the **NP**-complete cases lies, we refine the classical subproblems of the APS problem in much the same way as in [19] and prove that both APS($\{\sqsubset, \wr\}, \emptyset$) and APS($\{<, \wr\}, \emptyset$) are **NP**-complete. We end this paper by giving some new positive results, namely showing that APS($\{\wr\}$, $\emptyset$) and APS($\{\wr\}$,$\{\wr\}$) are polynomial time.

Keywords: RNA structures, Arc-Preserving Subsequence problem, Computational complexity.

1 Introduction

At a molecular state, the understanding of biological mechanisms is subordinated to the discovery and the study of RNA functions. Indeed, it is established that the

* This work was partially supported by the French-Italian PAI Galileo project number 08484VH and by the CNRS project ACI Masse de Données "NavGraphe". A preliminary version of this paper appeared in the Proc. of IWBRA'05, Springer, V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 860-868, 2005.

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 1–36, 2005.

conformation of a single-stranded RNA molecule (a linear sequence composed of ribonucleotides A, U, C and G, also called primary structure) partly determines the function of the molecule. This conformation results from the folding process due to local pairings between complementary bases ($A-U$ and $C-G$, connected by a hydrogen bond). The secondary structure of an RNA (a simplification of the complex 3-dimensional folding of the sequence) is the collection of folding patterns (stem, hairpin loop, bulge loop, internal loop, branch loop and pseudoknot) that occur in it.

RNA secondary structure comparison is important in many contexts, such as:

- identification of highly conserved structures during evolution, non detectable in the primary sequence which is often slightly preserved. These structures suggest a significant common function for the studied RNA molecules [16,18,13,8],
- RNA classification of various species (phylogeny)[4,3,21],
- RNA folding prediction by considering a set of already known secondary structures [24,14],
- identification of a consensus structure and consequently of a common role for molecules [22,5].

Structure comparison for RNA has thus become a central computational problem bearing many challenging computer science questions. At a theoretical level, the RNA structure is often modeled as an *arc-annotated sequence*, that is a pair (S, P) where S is the sequence of ribonucleotides and P represents the hydrogen bonds between pairs of elements of S. Different pattern matching and motif search problems have been investigated in the context of arc-annotated sequences among which we can mention the *arc-preserving subsequence* (APS) problem, the Edit Distance problem, the *arc-substructure* (AST) problem and the *longest arc-preserving subsequence* (LAPCS) problem (see for instance [6,15,12,11,2]). For other related studies concerning algorithmic aspects of (protein) structure comparison using *contact maps*, refer to [10,17].

In this paper, we focus on the *arc-preserving subsequence* (APS) problem: given two arc-annotated sequences (S, P) and (T, Q), this problem asks whether (T, Q) can be exactly obtained from (S, P) by deleting some of its bases together with their incident arcs, if any. This problem is commonly encountered when one is searching for a given RNA pattern in an RNA database [12]. Moreover, from a theoretical point of view, the APS problem can be seen as a restricted version of the LAPCS problem, and hence has applications in the structural comparison of RNA and protein sequences [6,10,23]. The APS problem has been extensively studied in the past few years [11,12,6]. Of course, different restrictions on arc-annotation alter the computational complexity of the APS problem, and hence this problem has been naturally divided into subproblems reflecting the complexity of the arc structure of both (S, P) and (T, Q): plain, chain, nested, crossing or unlimited (see Section 2 for details). All of them but one have been classified as to whether they are polynomial time solvable or **NP**-complete. The problem of the existence of a polynomial time algorithm for the APS(Crossing,Plain) problem was mentioned in [11] as the last open problem

Table 1. APS problem complexity where $n = |S|$ and $m = |T|$. $\star$ result from this paper.

APS				
	Crossing	Nested	Chain	Plain
Crossing	**NP**-complete [6]	**NP**-complete [12]		**NP**-complete $\star$
Nested		$O(nm)$ [11]		
Chain			$O(nm)$ [11]	$O(n+m)$ [11]

in the context of arc-preserving subsequences (cf. Table 1). Unfortunately, as we shall prove in Section 4, the APS(Crossing,Plain) problem is **NP**-complete even for restricted special cases.

In analyzing the computational complexity of a problem, we are often trying to define the precise boundary between the polynomial and the **NP**-complete cases. Therefore, as another step towards establishing the precise complexity landscape of the APS problem, it is of great interest to subdivide the existing cases into more precise ones, that is to refine the classical complexity levels of the APS problem, for determining more precisely what makes the problem hard. For that purpose, we use the framework introduced by Vialette [19] in the context of 2-intervals (a simple abstract structure for modelling RNA secondary structures). As a consequence, the number of complexity levels rises from 4 (not taking into account the unlimited case) to 8, and all the entries of this new complexity table need to be filled. Previous known results concerning the APS problem, along with two **NP**-completeness and two polynomiality proofs, allow us to fill all the entries of this new table, therefore determining what exactly makes the APS problem hard.

The paper is organized as follows. In Section 2, we give notations and definitions concerning the APS problem. In Section 3 we introduce and explain the new refinements of the complexity levels we are going to study. In Section 4, we show that the APS($\{\sqsubset, \between\}, \emptyset$) problem is **NP**-complete thereby proving that the (classical) APS(Crossing, Plain) problem is **NP**-complete as well. As another refinement to that result, we prove that the APS($\{<, \between\}, \emptyset$) problem is **NP**-complete. Finally, in Section 5, we give new polynomial time solvable algorithms for restricted instances of the APS(Crossing, Plain) problem.

2 Preliminaries

An RNA structure is commonly represented as an arc-annotated sequence (S, P) where S is the sequence of ribonucleotides (or bases) and P is the set of arcs connecting pairs of bases in S. Let (S, P) and (T, Q) be two arc-annotated sequences such that $|S| \geq |T|$ (in the following, $n = |S|$ and $m = |T|$). The APS problem asks whether (T, Q) can be exactly obtained from (S, P) by deleting some of its bases together with their incident arcs, if any.

Since the general problem is easily seen to be intractable [6], the arc structure must be restricted. Evans [6] proposed four possible restrictions on P (resp. Q) which were largely reused in the subsequent literature:

1. there is no base incident to more than one arc,
2. there are no arcs crossing,
3. there is no arc contained in another,
4. there is no arc.

These restrictions are used progressively and inclusively to produce five different levels of allowed arc structure:

- Unlimited - the general problem with no restrictions
- Crossing - restriction 1
- Nested - restrictions 1 and 2
- Chain - restrictions 1, 2 and 3
- Plain - restriction 4

Guo proved in [12] that the APS(Crossing, Chain) problem is **NP**-complete. Guo et al. observed in [11] that the **NP**-completeness of the APS(Crossing, Crossing) and APS(Unlimited, Plain) easily follows from results of Evans [6] concerning the LAPCS problem. Furthermore, they gave a $O(nm)$ time for the APS(Nested, Nested) problem. This algorithm can be applied to easier problems such as APS(Nested, Chain), APS(Nested, Plain), APS(Chain, Chain) and APS(Chain,Plain). Finally, Guo et al. mentioned in [11] that APS(Chain, Plain) can be solved in $O(n + m)$ time. Until now, the question of the existence of an exact polynomial algorithm for the problem APS(Crossing, Plain) remained open. We will first show in the present paper that the problem APS(Crossing,Plain) is **NP**-complete. Table 1 surveys known and new results for various types of APS. Observe that the Unlimited level has no restrictions, and hence is of limited interest in our study. Consequently, from now on we will not be concerned anymore with that level.

3 Refinement of the APS Problem

In this section, we propose a refinement of the APS problem. We first state formally our approach and explain why such a refinement is relevant for both theoretical and experimental studies. We end the section by giving easy properties of the proposed refinement that will prove extremely useful in Section 5.

3.1 Splitting the Levels

As we will show in Section 4, the APS(Crossing, Plain) problem is **NP**-complete. That result answers the last open problem concerning the computational complexity of the APS problem with respect to classical complexity levels, *i.e.*, Plain, Chain, Nested and Crossing (cf. Table 1). However, we are mainly interested in the elaboration of the precise border between **NP**-complete

and polynomially solvable cases. Indeed, both theorists and practitioners might naturally ask for more information concerning the hard cases of the APS problem in order to get valuable insight into what makes the problem difficult.

As a next step towards a better understanding of what makes the APS problem hard, we propose to refine the models which are classically used for classifying arc-annotated sequences. Our refinement consists in splitting those models of arc-annotated sequences into more precise relations between arcs. For example, such a refinement provides a general framework for investigating polynomial time solvable and hard restricted instances of APS(Crossing, Plain), thereby refining in many ways Theorem 1 (see Section 5).

We use the three relations first introduced by Vialette [19,20] in the context of 2-*intervals* (a simple abstract structure for modelling RNA secondary structures). Actually, his definition of 2-intervals could almost apply in this paper (the main difference lies in the fact that Vialette used 2-intervals for representing sets of contiguous arcs). Vialette defined three possible relations between 2-intervals that can be used for arc-annotated sequences as well. They are the following: for any two arcs $p_1 = (i, j)$ and $p_2 = (k, l)$ in P, we will write $p_1 < p_2$ if $i < j < k < l$ (*precedence* relation), $p_1 \sqsubset p_2$ if $k < i < j < l$ (*nested* relation) and $p_1 \between p_2$ if $i < k < j < l$ (*crossing* relation). Two arcs p_1 and p_2 are τ-comparable for some $\tau \in \{<, \sqsubset, \between\}$ if $p_1 \tau p_2$ or $p_2 \tau p_1$. Let $\mathcal{P}$ be a set of arcs and R be a non-empty subset of $\{<, \sqsubset, \between\}$. The set $\mathcal{P}$ is said to be *R-comparable* if any two distinct arcs of $\mathcal{P}$ are τ-comparable for some $\tau \in R$. An arc-annotated sequence (S, P) is said to be an R-arc-annotated sequence for some non-empty subset R of $\{<, \sqsubset, \between\}$ if P is R-comparable. We will write $R = \emptyset$ in case $P = \emptyset$. Observe that our model cannot deal with arc-annotated sequences which contain only one arc. However, having only one arc or none can not really affect the computational complexity of the problem. Just one guess reduces from one case to the other. Details are omitted here.

As a straightforward illustration of the above definitions, classical complexity levels for the APS problem can be expressed in terms of combinations of our new relations: Plain is fully described by $R = \emptyset$, Chain is fully described by $R = \{<\}$, Nested is fully described by $R = \{<, \sqsubset\}$ and Crossing is fully described by $R = \{<, \sqsubset, \between\}$. The key point is to observe that our refinement allows us to consider new structures for arc-annotated sequences, namely $R = \{\sqsubset\}$, $R = \{\between\}$, $R = \{<, \between\}$ and $R = \{\sqsubset, \between\}$, which could not be considered using the classical complexity levels. Although other refinements may be possible (in particular well-suited for parameterized complexity analysis), we do believe that such an approach allows a more precise analysis of the complexity of the APS problem.

Of course one might object that some of these subdivisions are unlikely to appear in RNA secondary structures. While this is true, it is also true that it is of great interest to answer, at least partly, the following question: Where is the precise boundary between the polynomial and the **NP**-complete cases? Indeed, such a question is relevant for both theoretical and experimental studies.

For one, many important optimization problems are known to be **NP**-complete. That is, unless $\mathbf{P} = \mathbf{NP}$, there is no polynomial time algorithm that optimally solves these on every input instance, and hence proving a problem to be **NP**-complete is generally accepted as a proof of its difficulty. However the problem to be solved may be much more specialized than the general one that was proved to be **NP**-complete. Therefore, during the past three decades, many studies have been devoted to proving **NP**-completeness results for highly restricted instances in order to precisely define the border between tractable and intractable problems. Our refinements have thus to be seen as another step towards establishing the precise complexity landscape of the APS problem.

For another, it is worthwhile keeping in mind that intractability must be coped with and problems must be solved in practical applications. Computer science theory has articulated a few general programs for systematically coping with the ubiquitous phenomena of computational intractability: average case analysis, approximation algorithm, randomized algorithm and fixed parameter complexity. Fully understanding where the boundary lies between efficiently solvable formulations and intractable ones is another important approach. Indeed, from an engineering point of view for which the emphasis is on efficiency, that precise boundary might be a good starting point for designing efficient heuristics or for exploring fixed-parameter tractability. The better our understanding of the problem, the better our ability in defining efficient algorithms for practical applications.

3.2 Immediate Results

First, observe that, as in Table 1, we only have to consider cases of APS(R_1,R_2) where R_1 and R_2 are compatible, i.e. $R_2 \subseteq R_1$. Indeed, if this is not the case, we can immediately answer negatively since there exists two arcs in T which satisfy a relation in R_2 which is not in R_1, and hence T simply cannot be obtained from S by deleting bases of S. Those incompatible cases are simply denoted by hatched areas in Table 2.

Table 2. Complexity results after refinement of the complexity levels. ////: incompatible cases. ?: open problems.

APS								
R_1 \ R_2	$\{<,\sqsubset,\emptyset\}$	$\{\sqsubset,\emptyset\}$	$\{<,\emptyset\}$	$\{\emptyset\}$	$\{<,\sqsubset\}$	$\{\sqsubset\}$	$\{<\}$	$\emptyset$
$\{<,\sqsubset,\emptyset\}$	**NP**-C [6]	?	**NP**-C [12]	?	**NP**-C [12]	?	**NP**-C [12]	?
$\{\sqsubset,\emptyset\}$		?	////	?	////	?	////	?
$\{<,\emptyset\}$			?	?	////	////	?	?
$\{\emptyset\}$				?	////	////	////	?
$\{<,\sqsubset\}$					$O(nm)$ [11]	$O(nm)$ [11]	$O(nm)$ [11]	$O(nm)$ [11]
$\{\sqsubset\}$						$O(nm)$ [11]	////	$O(nm)$ [11]
$\{<\}$							$O(nm)$ [11]	$O(n+m)$ [11]
$\emptyset$								$O(n+m)$ [11]

Some known results allow us to fill many entries of the new complexity table derived from our refinement. The remainder of this subsection is devoted to detailing these first easy statements. We begin with an observation concerning complexity propagation properties of the APS problems in our refined model.

Observation 1. *Let R_1, R_2, R'_1 and R'_2 be four subsets of $\{<, \sqsubset, \between\}$ such that $R'_2 \subseteq R_2 \subseteq R_1$ and $R'_2 \subseteq R'_1 \subseteq R_1$. If* $\mathrm{APS}(R'_1,\ R'_2)$ *is* **NP***-complete (resp.* $\mathrm{APS}(R_1,\ R_2)$ *is polynomial time solvable) then so is* $\mathrm{APS}(R_1,\ R_2)$ *(resp.* $\mathrm{APS}(R'_1,\ R'_2)$*).*

On the positive side, Gramm *et al.* have shown that APS(Nested, Nested) is solvable in $O(nm)$ time [11]. Another way of stating this is to say that $\mathrm{APS}(\{<, \sqsubset\},\ \{<, \sqsubset\})$ is solvable in $O(mn)$ time. That result together with Observation 1 may be summarized by saying that $\mathrm{APS}(R_1,\ R_2)$ for any compatible R_1 and R_2 such that $\between \notin R_1$ and $\between \notin R_2$ is polynomial time solvable.

Conversely, the **NP**-completeness of APS(Crossing,Crossing) has been proved by Evans [6]. A simple reading shows that her proof is concerned with $\{<, \sqsubset, \between\}$-arc-annotated sequences, and hence she actually proved that $\mathrm{APS}(\{<, \sqsubset, \between\},\ \{<, \sqsubset, \between\})$ is **NP**-complete. Similarly, in proving that APS(Crossing, Chain) is **NP**-complete [12], Guo actually proved that $\mathrm{APS}(\{<, \sqsubset, \between\},\ \{<\})$ is **NP**-complete. Note that according to Observation 1, this latter result implies that $\mathrm{APS}(\{<, \sqsubset, \between\},\ \{<, \sqsubset\})$ and $\mathrm{APS}(\{<, \sqsubset, \between\},\ \{<, \between\})$ are **NP**-complete.

Table 2 surveys known and new results for various types of our refined APS problem. Observe that this paper answers all questions concerning the APS problem with respect to the new complexity levels.

4 Hardness Results

We show in this section that $\mathrm{APS}(\{\sqsubset, \between\}, \emptyset)$ is **NP**-complete thereby proving that the (classical) APS(Crossing, Plain) problem is **NP**-complete. That result answers an open problem posed in [11], which was also the last open problem concerning the computational complexity of the APS problem with respect to classical complexity levels, *i.e.*, Plain, Chain, Nested and Crossing (cf. Table 1). Furthermore, we prove that the $\mathrm{APS}(\{<, \between\}, \emptyset)$ is **NP**-complete as well.

We provide a polynomial time reduction from the 3-Sat problem: Given a set $\mathcal{V}_n$ of n variables and a set $\mathcal{C}_q$ of q clauses (each composed of three literals) over $\mathcal{V}_n$, the problem asks to find a truth assignment for $\mathcal{V}_n$ that satisfies all clauses of $\mathcal{C}_q$. It is well-known that the 3-Sat problem is **NP**-complete [9].

It is easily seen that the $\mathrm{APS}(\{\sqsubset, \between\}, \emptyset)$ problem is in **NP**. The remainder of the section is devoted to proving that it is also **NP**-hard. Let $\mathcal{V}_n = \{x_1, x_2, ...x_n\}$ be a finite set of n variables and $\mathcal{C}_q = \{c_1, c_2, \ldots, c_q\}$ a collection of q clauses. Observe that there is no loss of generality in assuming that, in each clause, the literals are ordered from left to right, *i.e.*, if $c_i = (x_j \vee x_k \vee x_l)$ then $j < k < l$. Let us first detail the construction of the sequences S and T:

$$S = S^s_{x_1} A\ S^s_{\overline{x_1}}\ S^s_{x_2} A\ S^s_{\overline{x_2}} \dots S^s_{x_n} A\ S^s_{\overline{x_n}}\ S_{c_1}\ S_{c_2} \dots S_{c_q}\ S^e_{x_1}\ S^e_{x_2} \dots S^e_{x_n}$$
$$T = T^s_{x_1}\ T^s_{x_2} \dots T^s_{x_n}\ T_{c_1}\ T_{c_2} \dots T_{c_q}\ T^e_{x_1}\ T^e_{x_2} \dots T^e_{x_n}$$

We now detail the subsequences that compose S and T. Let γ_m (resp. $\gamma_{\overline{m}}$) be the number of occurrences of literal x_m (resp. $\overline{x_m}$) in $\mathcal{C}_q$ and let $k_m = \max(\gamma_m, \gamma_{\overline{m}})$. For each variable $x_m \in \mathcal{V}_n$, $1 \leq m \leq n$, we construct words $S^s_{x_m} = AC^{k_m}$, $S^s_{\overline{x_m}} = C^{k_m}A$ and $T^s_{x_m} = AC^{k_m}A$ where C^{k_m} represents a word of k_m consecutive bases C. For each clause c_i of $\mathcal{C}_q$, $1 \leq i \leq q$, we construct words $S_{c_i} = UGGGA$ and $T_{c_i} = UGA$. Finally, for each variable $x_m \in \mathcal{V}_n$, $1 \leq m \leq n$, we construct words $S^e_{x_m} = UUA$ and $T^e_{x_m} = UA$.

Having disposed of the two sequences, we now turn to defining the corresponding two arc structures (see Figure 1). In the following, $\mathcal{S}eq[i]$ will denote the i^{th} base of a sequence $\mathcal{S}eq$ and, for any $1 \leq m \leq n$, $l_{\overline{m}} = |S^s_{\overline{x_m}}|$. For all $1 \leq m \leq n$, we create the two following arcs: $(S^s_{x_m}[1], S^e_{x_m}[1])$ and $(S^s_{\overline{x_m}}[l_{\overline{m}}], S^e_{x_m}[2])$. For each clause c_i of $\mathcal{C}_q$, $1 \leq i \leq q$, and for each $1 \leq m \leq n$, if the k^{th} (*i.e.* 1^{st}, 2^{nd} or 3^{rd}) literal of c_i is x_m (resp. $\overline{x_m}$) then we create an arc between any free (i.e. not already incident to an arc) base C of $S^s_{\overline{x_m}}$ (resp. $S^s_{x_m}$) and the k^{th} base G of S_{c_i} (note that this is possible by definition of $S^s_{\overline{x_m}}$, $S^s_{x_m}$ and S_{c_i}). On the whole, the instance we have constructed is composed of $3q + 2n$ arcs. We denote by APS-CP-construction any construction of this type. In the following, we will distinguish arcs between bases A and U, denoted by AU-arcs, from arcs between bases C and G, denoted by CG-arcs. An illustration of an APS-CP-construction is given in Figure 1. Clearly, our construction can be carried out in polynomial time. Moreover, the result of such a construction is indeed an instance of $\mathrm{APS}(\{\sqsubset, \wr\}, \emptyset)$, since $Q = \emptyset$ (no arc is added to T) and P is a $\{\sqsubset, \wr\}$-comparable set (since there are no arcs $\{<\}$-comparable.

We begin by proving a canonicity lemma of an APS-CP-construction.

Lemma 1. *Let (S, P) and (T, Q) be any two arc-annotated sequences obtained from an* APS-CP*-construction. If (T, Q) can be obtained from (S, P) by deleting*

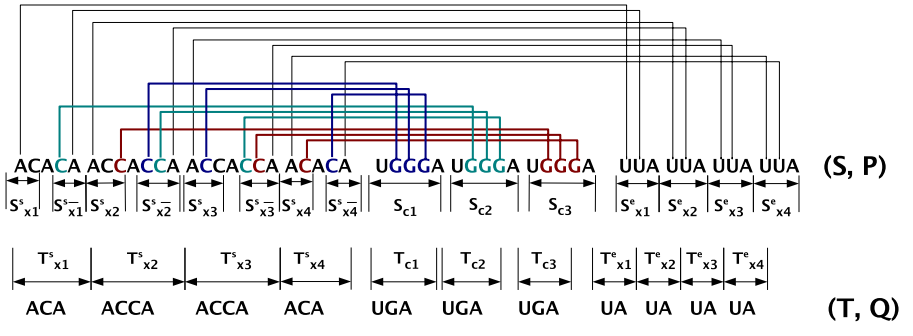


Fig. 1. Example of an APS-CP-construction with $\mathcal{C}_q = (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4})$

some of its bases together with their incident arcs, if any, then for each $1 \leq i \leq q$ *and* $1 \leq m \leq n$*:*

1. T_{c_i} *is obtained from* S_{c_i} *by deleting two of its three bases* G*,*
2. $T^e_{x_m}$ *is obtained from* $S^e_{x_m}$ *by deleting one of its two bases* U*,*
3. $T^s_{x_m}$ *is obtained from* $S^s_{x_m} A S^s_{\overline{x_m}}$ *by deleting either* $S^s_{x_m}$ *or* $S^s_{\overline{x_m}}$*.*

Proof. Let (S, P) and (T, Q) be two arc-annotated sequences resulting from an APS-CP-construction.
(1) By construction, the first base U appearing in S (resp. T) is $S_{c_1}[1]$ (resp. $T_{c_1}[1]$). Thus, $T_{c_1}[1]$ is obtained from a base U of S at, or after, $S_{c_1}[1]$. Moreover, the number of bases A appearing after $S_{c_1}[1]$ in S is equal to the number of bases A appearing after $T_{c_1}[1]$ in T. Therefore, every base A appearing after $S_{c_1}[1]$ and $T_{c_1}[1]$ must be matched. That is, for each $1 \leq i \leq q$, $T_{c_i}[3]$ is matched to $S_{c_i}[5]$. In particular, $T_{c_q}[3]$ is matched to $S_{c_q}[5]$. But since there are as many bases U between $S_{c_1}[1]$ and $S_{c_q}[5]$ as there are between $T_{c_1}[1]$ and $T_{c_q}[3]$, any base U in this interval in S must be matched to any base U in this interval in T; that is, for any $1 \leq i \leq q$, $T_{c_i}[1]$ is matched to $S_{c_i}[1]$. Thus, we conclude that for any $1 \leq i \leq q$, T_{c_i} is obtained by deleting two of the three bases G of S_{c_i}.
(2) By the above argument concerning the bases A appearing after $S_{c_1}[1]$ and $T_{c_1}[1]$, we know that if (T, Q) can be obtained from (S, P), then $T^e_{x_m}[2]$ is matched to $S^e_{x_m}[3]$ for any $1 \leq m \leq n$. Thus, for any $1 \leq m \leq n$, $T^e_{x_m}$ is obtained from $S^e_{x_m}$, and in particular $T^e_{x_m}[1]$ is matched to either $S^e_{x_m}[1]$ or $S^e_{x_m}[2]$.
(3) By definition, as there is no arc incident to bases of T, at least one base incident to every arc of P has to be deleted. We just mentioned that $T^e_{x_m}[1]$ is matched to either $S^e_{x_m}[1]$ or $S^e_{x_m}[2]$ for any $1 \leq m \leq n$. Thus, since by construction there is an arc between $S^e_{x_m}[1]$ and $S^s_{x_m}[1]$ (resp. $S^e_{x_m}[2]$ and $S^s_{\overline{x_m}}[l_{\overline{m}}]$), for any $1 \leq m \leq n$ either $S^s_{x_m}[1]$ or $S^s_{\overline{x_m}}[l_{\overline{m}}]$ has to be deleted; and all these arcs connect a base A appearing before $S_{c_1}[1]$ to a base U appearing after $S_{c_q}[5]$. Therefore, for any $1 \leq m \leq n$ a base A appearing before $S_{c_1}[1]$ in S is deleted. Originally, there are $3n$ bases A appearing before $S_{c_1}[1]$ in S and $2n$ appearing before the first base of $T_{c_1}[1]$ in T. Thus, the number of bases A matched in S and appearing before $S_{c_1}[1]$ is equal to the number of bases A appearing before $T_{c_1}[1]$ in T. But since, for each $1 \leq m \leq n$, a base A of either $S^s_{x_m}$ or $S^s_{\overline{x_m}}$ is deleted, we conclude that for each $1 \leq m \leq n$, $T^s_{x_m}$ is obtained from $S^s_{x_m} A S^s_{\overline{x_m}}$, by deleting either $S^s_{x_m}$ or $S^s_{\overline{x_m}}$. □

We now turn to proving that our construction is a polynomial time reduction from 3-Sat to APS(Crossing, Plain).

Lemma 2. *Let* I *be an instance of the problem* 3-Sat *with* n *variables and* q *clauses, and* I' *an instance* $((S, P); (T, Q))$ *of* APS$(\{\sqsubset, \emptyset\}, \emptyset)$ *obtained by an* APS-CP*-construction from* I*. An assignment of the variables that satisfies the boolean formula of* I *exists iff* T *is an Arc-Preserving Subsequence of* S*.*

Proof. ($\Rightarrow$) Suppose we have an assignment AS of the n variables that satisfies the boolean formula of I. By definition, for each clause there is at least one literal

that satisfies it. In the following, j_i will define, for any $1 \leq i \leq q$, the smallest index of the literal of c_i (i.e. 1, 2 or 3) which, by its assignment, satisfies c_i. Let (S, P) and (T, Q) be two sequences obtained from an APS-CP-construction from I. We look for a set $\mathcal{B}$ of bases to delete from S in order to obtain T. For each variable $x_m \in AS$ with $1 \leq m \leq n$, we define $\mathcal{B}$ as follows:

- if $x_m = True$ then $\mathcal{B}$ contains each base of $S^s_{\overline{x_m}}$ and $S^e_{x_m}[1]$,
- if $x_m = False$ then $\mathcal{B}$ contains each base of $S^s_{x_m}$ and $S^e_{x_m}[2]$,
- if $j_i = 1$ then $\mathcal{B}$ contains $S_{c_i}[3]$ and $S_{c_i}[4]$,
- if $j_i = 2$ then $\mathcal{B}$ contains $S_{c_i}[2]$ and $S_{c_i}[4]$,
- if $j_i = 3$ then $\mathcal{B}$ contains $S_{c_i}[2]$ and $S_{c_i}[3]$.

Since a variable has a unique value (i.e. $True$ or $False$), either each base of $S^s_{\overline{x_m}}$ and $S^e_{x_m}[1]$ or each base of $S^s_{x_m}$ and $S^e_{x_m}[2]$ are in $\mathcal{B}$ for all $1 \leq m \leq n$. Thus, $\mathcal{B}$ contains at least one base in S of any AU-arc of P.

For any $1 \leq i \leq q$, two of the three bases G of S_{c_i} are in $\mathcal{B}$. Thus, $\mathcal{B}$ contains at least one base in S of two thirds of the CG-arcs of P. Moreover, $S_{c_i}[j_i + 1]$ is the base G that is not in $\mathcal{B}$. We suppose in the following that the j_i^{th} literal of the clause c_i is x_m, with $1 \leq m \leq n$. Thus, by the way we build the APS-CP-construction, there is an arc between a base C of $S^s_{\overline{x_m}}$ and $S_{c_i}[j_i + 1]$ in P. By definition, if AS is an assignment of the n variables that satisfies the boolean formula, AS satisfies c_i and thus $x_m = True$. We mentioned, in the definition of $\mathcal{B}$ that if $x_m = True$ then each base of $S^s_{\overline{x_m}}$ is in $\mathcal{B}$. Thus, the base C of $S^s_{\overline{x_m}}$ incident to the CG-arc in P with $S_{c_i}[j_i+1]$ is in $\mathcal{B}$. A similar result can be found if the j_i^{th} literal of the clause c_i is $\overline{x_m}$. Thus, $\mathcal{B}$ contains at least one base in S of any CG-arc of P.

If S' is the sequence obtained from S by deleting all the bases of $\mathcal{B}$ together with their incident arcs, then there is no arc in S' (i.e. neither AU-arcs or CG-arcs). By the way we define $\mathcal{B}$, S' is obtained from S by deleting all the bases of either $S^s_{x_m}$ or $S^s_{\overline{x_m}}$, two bases G of S_{c_i} and either $S^e_{x_m}[1]$ or $S^e_{x_m}[2]$, for $1 \leq i \leq q$ and $1 \leq m \leq n$. According to Lemma 1, it is easily seen that sequence S' obtained is similar to T.

($\Leftarrow$) Let I be an instance of the problem 3-SAT with n variables and q clauses. Let I' be an instance $((S, P); (T, Q))$ of APS$(\{\sqsubset, \between\}, \emptyset)$ obtained by an APS-CP-construction from I such that (T, Q) can be obtained from (S, P) by deleting some of its bases (i.e. a set of bases $\mathcal{B}$) together with their incident arcs, if any. By Lemma 1, either all bases of $S^s_{\overline{x_m}}$ or all bases of $S^s_{x_m}$ are in $\mathcal{B}$. Consequently, for $1 \leq m \leq n$, we define an assignment AS of the n variables of I as follows:

- if all bases of $S^s_{\overline{x_m}}$ are in $\mathcal{B}$ then $x_m = True$,
- if all bases of $S^s_{x_m}$ are in $\mathcal{B}$ then $x_m = False$.

Now, let us prove that for any $1 \leq i \leq q$ the clause c_i is satisfied by AS. By Lemma 1, for any $1 \leq i \leq q$ there is a base G of substring S_{c_i} (say the $j_i + 1^{th}$) that is not in $\mathcal{B}$. By the the way we build the APS-CP-construction, there is a CG-arc in P between $S_{c_i}[j_i+1]$ and a base C of $S^s_{\overline{x_m}}$ (resp. $S^s_{x_m}$) if the j_i^{th} literal of c_i is x_m (resp. $\overline{x_m}$).

Suppose, *w.l.o.g.*, that the j_i^{th} literal of c_i is x_m. Since Q is an empty set, at least one base of any arc of P is in $\mathcal{B}$. Thus, the base C of $S^s_{\overline{x_m}}$ incident to the CG-arc in P with $S_{c_i}[j_i+1]$ is in $\mathcal{B}$ (since $S_{c_i}[j_i+1] \notin \mathcal{B}$). Therefore, by Lemma 1, all the bases of $S^s_{\overline{x_m}}$ are in $\mathcal{B}$. By the way we define AS, $x_m = True$ and thus c_i is satisfied. The same conclusion can be similarly derived if the j_i^{th} literal of c_i is $\overline{x_m}$. □

We have thus proved the following theorem.

Theorem 1. *The* APS($\{\sqsubset, \between\}, \emptyset$) *problem is* **NP***-complete.*

It follows immediately from Theorem 1 that the APS($\{<, \sqsubset, \between\}, \emptyset$) problem, and hence the classical APS(Crossing, Plain) problem, is **NP**-complete.

One might naturally ask for more information concerning the hard cases of the APS problem in order to get valuable insight into what makes the problem difficult. Another refinement of Theorem 1 is given by the following theorem.

Theorem 2. *The* APS($\{<, \between\}, \emptyset$) *problem is* **NP***-complete.*

As for Theorem 1, the proof is by reduction from the 3-Sat problem. It is easily seen that the APS($\{<, \between\}, \emptyset$) problem is in **NP**. The remainder of this section is devoted to proving that it is also **NP**-hard. Let $\mathcal{V}_n = \{x_1, x_2, ...x_n\}$ be a finite set of n variables and $\mathcal{C}_q = \{c_1, c_2, \ldots, c_q\}$ a collection of q clauses. The instance of the APS($\{<, \between\}, \emptyset$) problem we will build is decomposed in two parts: a *Truth Setting part* and a *Checking part*. For readability, we denote by APS2-cp-construction any construction of the type described hereafter. Moreover, we will present separately the *Truth Setting part* and the *Checking part*: first, we will describe the *Truth Setting part*, then the *Checking part* and end by the description of the set of arcs connecting those two parts. Indeed, the instance of the APS($\{<, \between\}, \emptyset$) problem will be the concatenation of those two parts.

Truth Setting part

Let us first detail the construction of sequences S' and T' of the *Truth Setting part*:

$$S' = \overbrace{\underbrace{S^e_{x_1}\ S^e_{x_2} \ldots S^e_{x_n}}_{T_{\alpha'}}}^{S_\alpha} \mathbf{GGG} \overbrace{\underbrace{S^s_{x_1} A\ S^s_{\overline{x_1}}\ S^s_{x_2} A\ S^s_{\overline{x_2}} \ldots S^s_{x_n} A\ S^s_{\overline{x_n}}}_{T_{\beta'}}}^{S_\beta}$$
$$T' = \underbrace{T^e_{x_1}\ T^e_{x_2} \ldots T^e_{x_n}}_{T_{\alpha'}} \mathbf{GGG} \underbrace{T^s_{x_1}\ T^s_{x_2} \ldots T^s_{x_n}}_{T_{\beta'}}$$

We now detail subsequences that compose S' and T'. Let γ_m (resp. $\gamma_{\overline{m}}$) be the number of occurrences of literal x_m (resp. $\overline{x_m}$) in $\mathcal{C}_q$ and let $k_m = \max(\gamma_m, \gamma_{\overline{m}})$. For each variable $x_m \in \mathcal{V}_n$, we construct substrings $S^e_{x_m} = UUA$, $T^e_{x_m} = UA$, $S^s_{x_m} = AC^{k_m}$, $S^s_{\overline{x_m}} = C^{k_m}A$ and $T^s_{x_m} = AC^{k_m}A$, where C^{k_m} represents a substring of k_m consecutive bases C. Having disposed of the two sequences, we now turn to defining the corresponding arc structure (see Figure 2). For all $1 \leq m \leq n$, we create the two following arcs: ($S^e_{x_m}[1]$,$S^s_{x_m}[1]$) and ($S^e_{x_m}[2]$,$S^s_{\overline{x_m}}[k_m + 1]$). Remark that, by now, all the arcs defined are $\{\between\}$-comparable.

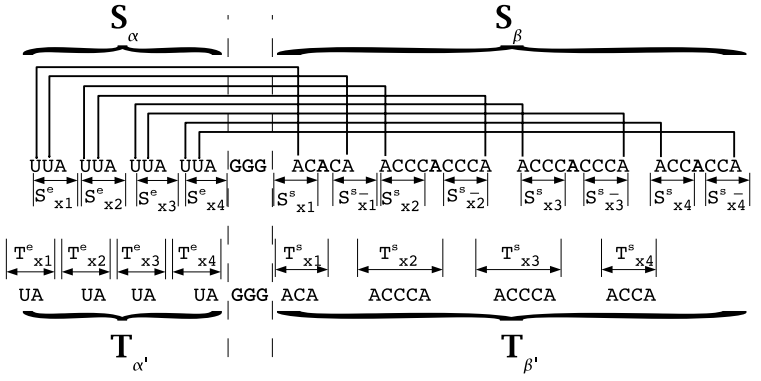


Fig. 2. The truth setting part of an APS2-CP-construction with $\mathcal{C}_q = (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4})$

Checking part

Let us now detail the construction of sequences S_ζ and $T_{\zeta'}$ of the *Checking part*:

$$S_\zeta = U \overbrace{S^1_{x_1} S^1_{x_2} ... S^1_{x_n}}^{S^1} U \overbrace{S^{\overline{1}}_{x_1} S^{\overline{1}}_{x_2} ... S^{\overline{1}}_{x_n}}^{S^{\overline{1}}} U ... U \overbrace{S^q_{x_1} S^q_{x_2} ... S^q_{x_n}}^{S^q} U \overbrace{S^{\overline{q}}_{x_1} S^{\overline{q}}_{x_2} ... S^{\overline{q}}_{x_n}}^{S^{\overline{q}}} U$$
$$T_{\zeta'} = U\ T^1\ U\ T^{\overline{1}}\ U ... U\ T^q\ U\ T^{\overline{q}}\ U$$

We now detail subsequences that compose S_ζ and $T_{\zeta'}$. For any $1 \leq m \leq n$ and any $1 \leq i \leq q$, let γ^i_m (resp. $\gamma^i_{\overline{m}}$) be the number of occurrences of literal x_m (resp. $\overline{x_m}$) in the set of clauses c_j with $i < j \leq q$ and let $\lambda^i_m = \gamma^i_m + \gamma^i_{\overline{m}}$. For any $1 \leq m \leq n$ and for any $1 \leq i \leq q$, let $y^i_m = 1$ if $x_m \in c_i$, $y^i_m = 0$ otherwise. For any $1 \leq m \leq n$ and for any $1 \leq i \leq q$, let $y^i_{\overline{m}} = 1$ if $\overline{x_m} \in c_i$, $y^i_{\overline{m}} = 0$ otherwise. For any $1 \leq m \leq n$ and $1 \leq i \leq q$, we construct substrings:

$$S^i_{x_m} = (GGA)^{\lambda^i_m + y^i_{\overline{m}}} (GA)^{y^i_m} (GGA)^{\lambda^i_m + y^i_m} (GA)^{y^i_{\overline{m}}}$$
$$S^{\overline{i}}_{x_m} = (CCA)^{\lambda^i_m} (CA)^{y^i_{\overline{m}}} (CCA)^{\lambda^i_m} (CA)^{y^i_m}$$
$$T^i = (GA)^{4+6q-6i}$$
$$T^{\overline{i}} = (CA)^{2+6q-6i}$$

For example, assuming that $\mathcal{C}_q = (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4})$ we have, among others, the following segments:

$$S^1_{x_1} = (GGA)^1 (GA)^0 (GGA)^1 (GA)^0 = GGA\ GGA$$
$$S^1_{x_2} = (GGA)^2 (GA)^1 (GGA)^3 = GGA\ GGA\ GA\ GGA\ GGA\ GGA$$
$$S^{\overline{2}}_{x_3} = (CCA)^1 (CA)^0 (CCA)^1 (CA)^1 = CCA\ CCA\ CA$$

$$T^2 = (GA)^{4+6*3-6*2} = GA\ GA\ GA\ GA\ GA\ GA\ GA\ GA\ GA\ GA$$
$$T^{\overline{3}} = (CA)^{2+6*3-6*3} = CA\ CA$$

Having disposed of the two sequences, we now turn to defining the corresponding arc structure (see Figure 3). By construction, $S^i_{x_m}$ (resp. $S^{\overline{i}}_{x_m}$) is composed of substrings GA and GGA (resp. CA and CCA). We denote by *repeater* any substring GGA or CCA. We denote by *terminal* any substring GA or CA which is not part of a repeater. Let $term(i, m, j)$ (resp. $rep(i, m, j)$) be the j^{th} terminal (resp. repeater) of $S^i_{x_m}$, and let $term(\overline{i}, m, j)$ (resp. $rep(\overline{i}, m, j)$) be the j^{th} terminal (resp. repeater) of $S^{\overline{i}}_{x_m}$.

For all $1 \leq m \leq n$, $1 \leq j \leq 2\lambda^i_m + 1$ and $1 \leq i < q$, we create the following arcs:

- an arc between the second base G of $rep(i, m, j)$ and the first base C of the j^{th} element (i.e. either a terminal or a repeater) of $S^{\overline{i}}_{x_m}$;
- an arc between the second base C of $rep(\overline{i}, m, j)$ and the first base G of the j^{th} element of $S^{i+1}_{x_m}$.

Final Construction

Final sequences S and T are respectively obtained by concatenating S' with S_ζ and T' with $T_{\zeta'}$. Moreover, we create, for all $1 \leq m \leq n$ and all $1 \leq j \leq \gamma_m + \gamma_{\overline{m}}$, an arc between the j^{th} base C of substring $S^s_{x_m} AS^s_{\overline{x_m}}$ in S' and the first base G of the j^{th} element of $S^1_{x_m}$ in S_ζ. In the rest of the paper, S^i will refer to $S^i_{x_1}\ S^i_{x_2}\ \ldots\ S^i_{x_n}$ and $S^{\overline{i}}$ will refer to $S^{\overline{i}}_{x_1}\ S^{\overline{i}}_{x_2} \ldots S^{\overline{i}}_{x_n}$.

In the following, we will show that P is $\{<, \between\}$-comparable. Let a_1 and a_2 be any two arcs connecting a base of S_β to a base of S_ζ. As all the arcs connecting a base of S_β to a base of S_ζ are of the same form, we consider, w.l.o.g. that:

- for a given j and a given $1 \leq m \leq n$, a_1 is the arc which connects the j^{th} base C of substring $S^s_{x_m} AS^s_{\overline{x_m}}$ to the first base G of the j^{th} element of $S^1_{x_m}$;
- for a given k and a given $1 \leq m' \leq n$, a_2 is the arc which connects the k^{th} base C of substring $S^s_{x_{m'}} AS^s_{\overline{x_{m'}}}$ to the first base G of the k^{th} element of $S^1_{x_{m'}}$;
- $j < k$.

We now consider the three following cases: (i) $m = m'$, (ii) $m < m'$ and (iii) $m > m'$. Suppose $m = m'$. As $j < k$, the j^{th} base C precedes the k^{th} base C of substring $S^s_{x_m} AS^s_{\overline{x_m}}$. Moreover, the first base G of the j^{th} element of $S^1_{x_m}$ precedes the first base G of the k^{th} element of $S^1_{x_m}$. Thus, a_1 and a_2 are $\{\between\}$-comparable.

Suppose now $m < m'$. Then, the j^{th} base C of substring $S^s_{x_m} AS^s_{\overline{x_m}}$ precedes the k^{th} base C of substring $S^s_{x_{m'}} AS^s_{\overline{x_{m'}}}$. Moreover, the first base G of the j^{th} element of $S^1_{x_m}$ precedes the first base G of the k^{th} element of $S^1_{x_{m'}}$. Thus, a_1 and a_2 are $\{\between\}$-comparable. The case where $m > m'$ is fully similar. Therefore,

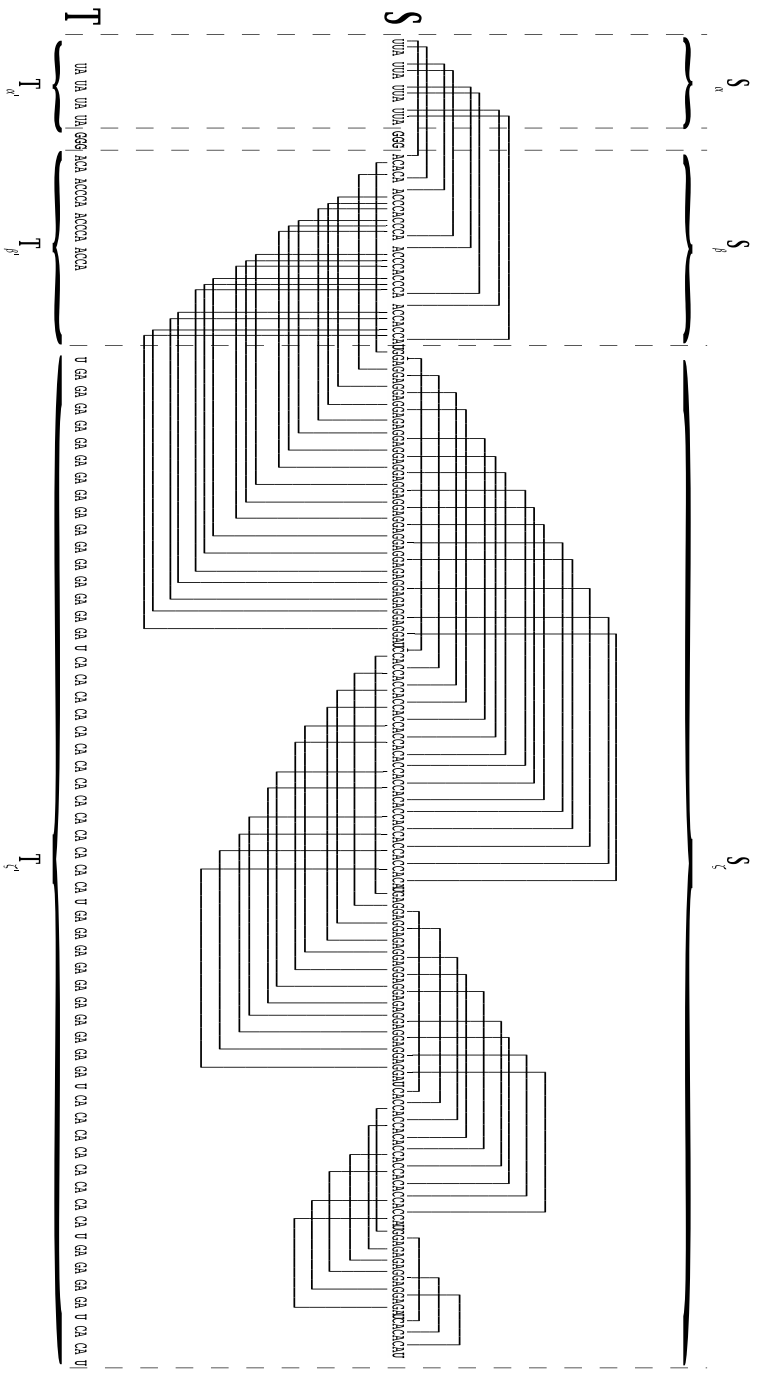


Fig. 3. Example of an APS2-CP-construction with $\mathcal{C}_q = (x_2 \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee \overline{x_4})$

given two arcs a_1 and a_2 connecting a base of S_β and a base of S_ζ, a_1 and a_2 are $\{\between\}$-comparable, and thus, $\{<, \between\}$-comparable.

Let a_1 and a_2 be any two arcs connecting two bases of S_ζ. There are two types of arcs connecting two bases of S_ζ:

1. arcs connecting, for a given $1 \leq i \leq q$ and a given j, a base of the j^{th} repeater of S^i to a base of the j^{th} element of $S^{\overline{i}}$;
2. arcs connecting, for a given $1 \leq i < q$ and a given j, a base of the j^{th} repeater of $S^{\overline{i}}$ to a base of the j^{th} element of S^{i+1}.

By definition, a_1 and a_2 can be either of type 1 or type 2. Since the cases where a_1 and a_2 are of different types are fully similar, we detail hereafter three cases: (a) a_1 and a_2 are of type 1, (b) a_1 is of type 1 and a_2 is of type 2, and (c) a_1 and a_2 are of type 2.

(a) Suppose that a_1 and a_2 are of type 1. Since a_2 is of type 1, a_2 connects, for a given $1 \leq i' \leq q$ and a given k, a base of the k^{th} repeater of $S^{i'}$ to a base of the k^{th} element of $S^{\overline{i'}}$. Suppose, w.l.o.g., that $j < k$. By construction, if $i \neq i'$ then either a_1 precedes a_2 or a_2 precedes a_1. Therefore, if $i \neq i'$ then a_1 and a_2 are $\{<\}$-comparable. Moreover, if $i = i'$ then a_1 and a_2 are $\{\between\}$-comparable.

(b) Suppose that a_1 is of type 1 and a_2 is of type 2. Since a_2 is of type 2, a_2 connects, for a given $1 \leq i' \leq q$ and a given k, a base of the k^{th} repeater of $S^{\overline{i'}}$ to a base of the k^{th} element of $S^{i'+1}$. By construction, if $i \neq i'$ then either a_1 precedes a_2 or a_2 precedes a_1. Therefore, if $i \neq i'$ then a_1 and a_2 are $\{<\}$-comparable. Consider now the case where $i = i'$. Suppose first that $j < k$. If $i = i'$ then, as $S^{\overline{i}}$ precedes S^{i+1} and $j < k$, a_1 and a_2 are $\{<\}$-comparable. Suppose now that $j > k$. If $i = i'$ then, as $S^{\overline{i}}$ precedes S^{i+1} and $k < j$, a_1 and a_2 are $\{\between\}$-comparable.

(c) Suppose that a_1 and a_2 are of type 2. Since a_2 is of type 2, a_2 connects, for a given $1 \leq i' \leq q$ and a given k, a base of the k^{th} repeater of $S^{\overline{i'}}$ to a base of the k^{th} element of $S^{i'+1}$. Suppose, w.l.o.g., that $j < k$. By construction, if $i \neq i'$ then either a_1 precedes a_2 or a_2 precedes a_1. Therefore, if $i \neq i'$ then a_1 and a_2 are $\{<\}$-comparable. Moreover, if $i = i'$ then a_1 and a_2 are $\{\between\}$-comparable.

Therefore, given two arcs a_1 and a_2 connecting two bases of S_ζ, a_1 and a_2 are $\{<, \between\}$-comparable. We now turn to proving that the set P is $\{<, \between\}$-comparable. Notice, first, that there is no arc connecting two bases of S_β (resp. S_α). We proved previously that given two arcs a_1 and a_2 connecting a base of S_β and a base of S_ζ, a_1 and a_2 are $\{<, \between\}$-comparable. Finally, we proved that given two arcs a_1 and a_2 connecting a base of S_α and a base of S_β, a_1 and a_2 are $\{\between\}$-comparable.Therefore, the set of arcs starting in $S_\alpha \bigcup S_\beta$ is $\{<, \between\}$-comparable.

Let $a_\zeta = (u', v')$, where u' and v' are bases, denote the arc connecting a base of S_β to a base of S_ζ and which ends the last. By construction, all the arcs connecting two bases of S_ζ are ending after v'. Therefore, the set of arcs in S (*i.e.* the set P) is $\{<, \between\}$-comparable.

A full illustration of an APS2-CP-construction is given in Figure 3. Clearly, our construction can be carried out in polynomial time. Moreover, the result of such a construction is indeed an instance of APS($\{<, \between\}, \emptyset$), since $Q = \emptyset$ (no arc is added to T) and P is a $\{<, \between\}$-comparable set of arcs.

Let (S, P) and (T, Q) be two sequences obtained from an APS2-CP-construction. In the following, we will give some technical lemmas that will be useful for the comprehension of proof of Theorem 2.

Definition 1. *A canonical alignment of two sequences (S, P) and (T, Q) obtained from an* APS2-CP*-construction is an alignment where, for any $1 \leq i \leq q$ and $1 \leq m \leq n$:*

- *any base of $S^e_{x_m}$ is either matched with a base of $T^e_{x_m}$ or deleted,*
- *either each base of $S^s_{x_m}A$ is matched with a base of $T^s_{x_m}$ and all bases of $S^s_{\overline{x_m}}$ are deleted, or each base of $AS^s_{\overline{x_m}}$ is matched with a base of $T^s_{x_m}$ and all bases of $S^s_{x_m}$ are deleted,*
- *any base of S^i is either matched with a base of T^i or deleted,*
- *any base of $S^{\overline{i}}$ is either matched with a base of $T^{\overline{i}}$ or deleted.*

Lemma 3. *Let (S, P) and (T, Q) be two sequences obtained from an* APS2-CP*-construction. If (T, Q) is an arc-preserving subsequence of (S, P) then any corresponding alignment is canonical.*

Proof. Suppose (T, Q) is an arc-preserving subsequence of (S, P). Let $\mathcal{A}$ denote any corresponding alignment. In T, there is a substring GGG between $T_{\alpha'}$ and $T_{\beta'}$. In S, bases G are present either between S_α and S_β, or in S_ζ. The number of bases U in S_ζ and in $T_{\zeta'}$ is equal. Moreover, in both S_ζ and $T_{\zeta'}$ the first (*i.e.* leftmost) base is a base U. Therefore, in $\mathcal{A}$, none of the bases of the substring GGG in T between $T_{\alpha'}$ and $T_{\beta'}$ can be matched to a base G of S_ζ since, in that case, at least one base U of $T_{\zeta'}$ would not be matched. Thus, in $\mathcal{A}$, substring GGG of S has to be matched with substring GGG of T and $T_{\alpha'}$ must be matched with substrings of S_α.

Moreover, the number of bases U in S_ζ and in $T_{\zeta'}$ is equal; besides, in S_β and $T_{\beta'}$ there is no base U. Thus, $T_{\beta'}$ (resp. $T_{\zeta'}$) must be matched with substrings of S_β (resp. S_ζ). Therefore, we will consider the three cases ($S_\alpha/T_{\alpha'}$, $S_\beta/T_{\beta'}$, $S_\zeta/T_{\zeta'}$) separately.

Consider S_α and $T_{\alpha'}$. There are exactly n bases A both in S_α and $T_{\alpha'}$. Consequently, in $\mathcal{A}$, for all $1 \leq m \leq n$, $S^e_{x_m}$ has to be matched with $T^e_{x_m}$. More precisely, $T^e_{x_m}[1]$ has to be matched to either $S^e_{x_m}[1]$ or $S^e_{x_m}[2]$ for all $1 \leq m \leq n$.

Consider S_β and $T_{\beta'}$. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. We just mentioned that $T^e_{x_m}[1]$ has to be matched to either $S^e_{x_m}[1]$ or $S^e_{x_m}[2]$ for any $1 \leq m \leq n$. Thus, since by construction there is an arc between $S^e_{x_m}[1]$ and $S^s_{x_m}[1]$ (resp. $S^e_{x_m}[2]$ and $S^s_{\overline{x_m}}[k_m + 1]$), for any $1 \leq m \leq n$, either $S^s_{x_m}[1]$ or $S^s_{\overline{x_m}}[k_m + 1]$ is deleted. Therefore, n bases A appearing in S_β are deleted. Note that there are $3n$ bases A in S_β and $2n$ in $T_{\beta'}$. Thus, the number of bases A not deleted in S_β is equal to the number of bases A in $T_{\beta'}$. Since, for each $1 \leq m \leq n$, a base A of either $S^s_{x_m}$ or $S^s_{\overline{x_m}}$ is

deleted, we conclude that for each $1 \leq m \leq n$, $T^s_{x_m}$ is obtained from $S^s_{x_m} A S^s_{\overline{x_m}}$, by deleting all bases of either $S^s_{x_m}$ or $S^s_{\overline{x_m}}$.

Consider S_ζ and $T_{\zeta'}$. By construction, there are $2q+1$ bases U in S_ζ and in $T_{\zeta'}$. Thus, in $\mathcal{A}$, the $2q+1$ bases U of S_ζ have to be matched with the $2q+1$ bases U of $T_{\zeta'}$. Therefore, in $\mathcal{A}$, for any $1 \leq i \leq q$, any base of S^i is either matched with a base of T^i or deleted, and any base of $S^{\overline{i}}$ is either matched with a base of $T^{\overline{i}}$ or deleted. □

In the following, given an alignment $\mathcal{A}$ of S and T, if the first base of a terminal is matched (resp. deleted) in $\mathcal{A}$ then the corresponding terminal will be denoted as *active* (resp. *inactive*). Similarly, a repeater is said to be inactive (resp. active) when its two first bases (resp. exactly one out of its two first bases) are deleted in $\mathcal{A}$. Notice that the case where none of the two first bases of a repeater is deleted in $\mathcal{A}$ is not considered.

Notice that, by construction, for any $1 \leq i \leq q$, there are no two consecutive bases G in $T_{\zeta'}$, and there are no two consecutive bases C in $T_{\zeta'}$. Thus, at least one out of any two consecutive bases C or G of S_ζ is deleted in $\mathcal{A}$. Therefore, given a canonical alignment, for any repeater of S, either the repeater is active or all its bases C or G are deleted.

Lemma 4. *Let (S, P) and (T, Q) be two sequences obtained from an* APS2-CP-*construction. If (T, Q) is an arc-preserving subsequence of (S, P), then for any corresponding alignment $\mathcal{A}$ and for any $1 \leq i \leq q$, one of the three following cases must occur:*

- *all the repeaters and one terminal of S^i are active,*
- *all the repeaters but one and two terminals of S^i are active,*
- *all the repeaters but two and three terminals of S^i are active.*

Proof. By Lemma 3, $\mathcal{A}$ is canonical. Moreover, by definition, in any canonical alignment, for all $1 \leq i \leq q$, any base of S^i is either matched with a base of T^i or deleted. Let ω_j (resp. ω'_j) denote the j^{th} element of S^i (resp. T^i).

By construction, in T^i, there are two bases A less than in S^i. Therefore, we know that in $\mathcal{A}$, all the bases A of S^i but two will be matched. Let ω_k and ω_l, with $k < l$, denote the two elements of S^i which contain the deleted bases A. There are two cases, as illustrated in Figure 4: either (a) $l = k+1$ or (b) $l > k+1$. Let us consider those two cases separately.

(a) Suppose $l = k+1$ (*i.e.* ω_k and ω_l are consecutive). In that case, since all the bases A but two will be matched in S^i, the base A of ω_{k-1} (resp. ω_{l+1}) is matched with a base A of an element of T^i, say ω'_m (resp. ω'_{m+1}). Therefore, the base G of ω'_{m+1} is either matched with a base of ω_k, ω_l or ω_{l+1}. In each of those cases, all the elements but two of S^i are active.

(b) Suppose $l > k+1$ (*i.e.* ω_k and ω_l are not consecutive). In that case, since all the bases A but two will be matched in S^i, the base A of ω_{k-1} (resp. ω_{k+1}) is matched with a base A of an element of T^i, say ω'_m (resp. ω'_{m+1}). Similarly, the base A of ω_{l-1} (resp. ω_{l+1}) is matched with a base A of an element of T^i, say ω'_p (resp. ω'_{p+1}). Therefore, the base G of ω'_{m+1} (resp. ω'_{p+1}) is either matched with

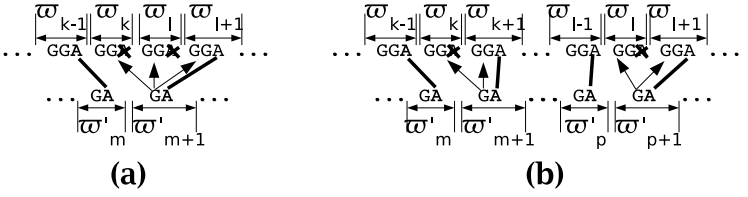


Fig. 4. Illustration of Lemma 4. (a) $l = k + 1$ or (b) $l > k + 1$.

a base of ω_k or ω_{k+1} (resp. ω_l or ω_{l+1}). In each of those cases, all the elements but two of S^i are active.

Therefore, either two terminals, or one repeater and one terminal, or two repeaters of S^i are inactive. □

Lemma 5. *Let (S, P) and (T, Q) be two sequences obtained from an* APS2-CP*-construction. If (T, Q) is an arc-preserving subsequence of (S, P), then for any corresponding alignment $\mathcal{A}$, all the repeaters and two terminals of $S^{\overline{1}}$ are active.*

Proof. Note that in this lemma, we focus on the first clause (*i.e.* c_1). c_1 is defined by three literals (say x_i, x_j and x_k). Since c_1 is equal to the disjunction of variables built with x_i, x_j and x_k, c_1 can have eight different forms, because each literal can appear in either its positive (x_i) or negative ($\overline{x_i}$) form. In the following, we suppose, to illustrate the proof, that $c_1 = (x_i \vee x_j \vee \overline{x_k})$ as illustrated in Figure 5. The other cases will not be considered here, but can be treated similarly.

By Lemma 3, $\mathcal{A}$ is canonical. Moreover, by definition, in any canonical alignment, for all $1 \leq i \leq q$, any base of $S^{\overline{i}}$ is either matched with a base of $T^{\overline{i}}$ or deleted. We recall that ω_j (resp. ω'_j) denotes the j^{th} element of $S^{\overline{i}}$ (resp. $T^{\overline{i}}$).

By construction, in $T^{\overline{1}}$, there is one base A less than in $S^{\overline{1}}$. Therefore, we know that in $\mathcal{A}$, all the bases A of $S^{\overline{1}}$ but one will be matched. Let ω_k denote the element of $S^{\overline{1}}$ which contains the deleted base A. Since all the bases A of $S^{\overline{1}}$ but two will be matched, the base A of ω_{k-1} (resp. ω_{k+1}) is matched with a base A of an element of $T^{\overline{1}}$, say ω'_m (resp. ω'_{m+1}). Therefore, the base C of ω'_{m+1} is either matched with a base of ω_k or ω_{k+1}. Consequently, all the elements but one of $S^{\overline{1}}$ are active.

To prove that the inactive element is a terminal, we suppose, by contradiction, that one repeater of $S^{\overline{1}}$ is inactive. Therefore, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active. Moreover, by Lemma 4, either:

1. all the repeaters of S^1 and one terminal of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active,
2. all the repeaters but one of S^1 and two terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active,
3. all the repeaters but two of S^1 and three terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active.

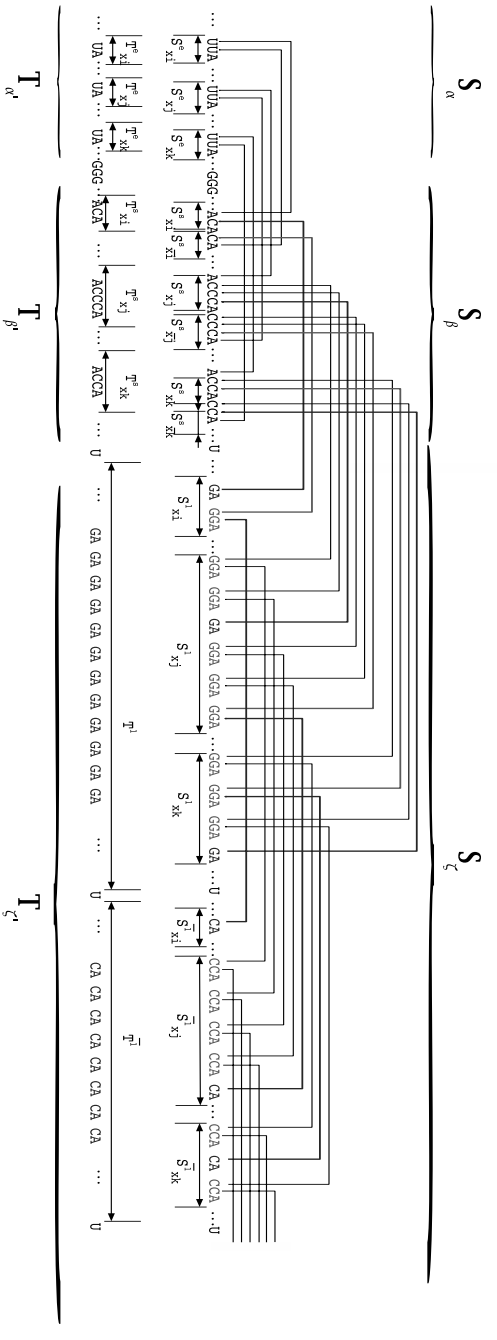


Fig. 5. Part of an APS2-CP-construction corresponding to a clause $c_1 = (x_i \vee x_j \vee \overline{x_k})$. Bold arcs correspond to the different cases studied in Lemma 5.

Let us consider those three cases separately:

(1) Suppose that all the repeaters of S^1 and one terminal of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. The active terminal can be in either $S^1_{x_i}$, $S^1_{x_j}$ or $S^1_{x_k}$. We recall that the clause considered is $c_1 = (x_i \vee x_j \vee \overline{x_k})$. Since the cases where the active terminal is either in $S^1_{x_i}$ or $S^1_{x_j}$ are fully similar, we detail hereafter only two cases: (a) the active terminal is in $S^1_{x_i}$ and (b) the active terminal is in $S^1_{x_k}$.
   (a) Suppose that the active terminal is in $S^1_{x_i}$. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$, $(rep[2], \theta) \in P$ where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Since, by hypothesis, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active, then θ is matched. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. Therefore, $rep[2]$ is deleted. Since rep is an active repeater, $rep[1]$ is matched. Thus, δ is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{x_i}$ and the first base of the terminal in $S^1_{x_i}$ (cf. Figure 5). Therefore, since the first base of terminal in $S^1_{x_i}$ is matched (because we supposed that the active terminal is in $S^1_{x_i}$), a base C of $S^s_{x_i}$ is deleted. Thus, a base of both $S^s_{x_i}$ and $S^s_{\overline{x_i}}$ is deleted. Therefore, by Definition 1, the alignment is not canonical, a contradiction.
   (b) Suppose now that the active terminal is in $S^1_{x_k}$. By construction, there is a repeater rep of $S^1_{x_k}$ such that $(\delta, rep[1]) \in P$, $(rep[2], \theta) \in P$ where δ (resp. θ) is a base C of $S^s_{x_k}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_k}$), as illustrated in Figure 5. Since, by hypothesis, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active, then θ is matched. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. Therefore, $rep[2]$ is deleted. Since rep is an active repeater, $rep[1]$ is matched. Thus, δ is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{\overline{x_k}}$ and the first base of the terminal in $S^1_{x_k}$ (cf. Figure 5). Therefore, since the first base of terminal in $S^1_{x_k}$ is matched (because we supposed that the active terminal is in $S^1_{x_k}$), a base C of $S^s_{\overline{x_k}}$ is deleted. Thus, a base of both $S^s_{x_k}$ and $S^s_{\overline{x_k}}$ is deleted. Therefore, by Definition 1, the alignment is not canonical, a contradiction.

(2) Suppose that all the repeaters but one of S^1 and two terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. The active terminals can be in either $(S^1_{x_i}, S^1_{x_j})$, $(S^1_{x_i}, S^1_{x_k})$ or $(S^1_{x_j}, S^1_{x_k})$. Since the cases where the active terminals are either in $(S^1_{x_i}, S^1_{x_k})$ or $(S^1_{x_j}, S^1_{x_k})$ are fully similar, we detail hereafter only two cases: (a) the active terminals are in $(S^1_{x_i}, S^1_{x_j})$ and (b) the active terminals are in $(S^1_{x_i}, S^1_{x_k})$.
   (a) Suppose that the active terminals are in $(S^1_{x_i}, S^1_{x_j})$. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$, $(rep[2], \theta) \in P$ where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Similarly, by construction, there is a repeater rep' of $S^1_{x_j}$ such that $(\delta', rep'[1]) \in P$, $(rep'[2], \theta') \in P$ where δ' (resp. θ') is a base C of $S^s_{\overline{x_j}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_j}$).

Since, by hypothesis, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active, then θ and θ' are matched. Therefore, since both $\{\theta, \theta'\}$ are matched, $rep[2]$ and $rep'[2]$ are deleted. Since either rep or rep' is active, either $rep[1]$ or $rep'[1]$ is matched. Thus, either δ or δ' is deleted.

Moreover, by construction, there is an arc between a base C of $S^s_{x_i}$ (resp. $S^s_{x_j}$) and the first base of the terminal in $S^1_{x_i}$ (resp. $S^1_{x_j}$). Therefore, since two terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active, at least one base C of either $S^s_{x_i}$ or $S^s_{x_j}$ is deleted. Thus, a base of either both $S^s_{x_i}$ and $S^s_{\overline{x_i}}$ or both $S^s_{x_j}$ and $S^s_{\overline{x_j}}$ is deleted. Consequently, by Definition 1, the alignment is not canonical, a contradiction.

(b) Suppose now that the active terminals are in $(S^1_{x_i}, S^1_{x_k})$. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$, $(rep[2], \theta) \in P$ where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Similarly, by construction, there is a repeater rep' of $S^1_{x_k}$ such that $(\delta', rep'[1]) \in P$, $(rep'[2], \theta') \in P$ where δ' (resp. θ') is a base C of $S^s_{x_k}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_k}$). Since, by hypothesis, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active, then θ and θ' are matched. Therefore, since both $\{\theta, \theta'\}$ are matched, $rep[2]$ and $rep'[2]$ are deleted. Since either rep or rep' is active, either $rep[1]$ or $rep'[1]$ is matched. Thus, either δ or δ' is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{x_i}$ (resp. $S^s_{\overline{x_k}}$) and the first base of the terminal in $S^1_{x_i}$ (resp. $S^1_{x_k}$). Therefore, since two terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active, at least one base C of either $S^s_{x_i}$ or $S^s_{\overline{x_k}}$ is deleted. Thus, a base of either both $S^s_{x_i}$ and $S^s_{\overline{x_i}}$ or both $S^s_{x_k}$ and $S^s_{\overline{x_k}}$ is deleted. Consequently, by Definition 1, the alignment is not canonical, a contradiction.

(3) Suppose that all the repeaters but two of S^1 and three terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. By construction, there is a repeater rep such that $(\delta, rep[1]) \in P$, $(rep[2], \theta) \in P$ where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$). Similarly, by construction, there is a repeater rep' such that $(\delta', rep'[1]) \in P$, $(rep'[2], \theta') \in P$ where δ' (resp. θ') is a base C of $S^s_{\overline{x_j}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_j}$). By construction, there is a repeater rep'' such that $(\delta'', rep''[1]) \in P$, $(rep''[2], \theta'') \in P$ where δ'' (resp. θ'') is a base C of $S^s_{x_k}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_k}$). Since, by hypothesis, the three terminals of $\{S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}\}$ are active, then θ, θ' and θ'' are matched. Therefore, since both $\{\theta, \theta', \theta''\}$ are matched, $rep[2]$, $rep'[2]$ and $rep''[2]$ are deleted. Since either rep, rep' or rep'' is active, either $rep[1]$, $rep'[1]$ or $rep''[1]$ is matched. Thus, either δ, δ' or δ'' is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{x_i}$ (resp. $S^s_{x_j}$ and $S^s_{\overline{x_k}}$) and the first base of the terminal in $S^1_{x_i}$ (resp. $S^1_{x_j}$ and $S^1_{x_k}$). Therefore, since three terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active, at least one base C of either $S^s_{x_i}$, $S^s_{x_j}$ or $S^s_{\overline{x_k}}$ is deleted. Thus, a base of either both $S^s_{x_i}$ and $S^s_{\overline{x_i}}$ or both $S^s_{x_j}$ and $S^s_{\overline{x_j}}$ or both $S^s_{x_k}$ and $S^s_{\overline{x_k}}$ is deleted. Therefore, by Definition 1, the alignment is not canonical, a contradiction.

Thus, the hypothesis that one repeater of $S^{\overline{1}}$ is inactive is wrong. Consequently, only a terminal of $S^{\overline{1}}$ can be inactive. We deduce that all the repeaters and two terminals of $S^{\overline{1}}$ are active. □

We now turn to proving that our construction is a polynomial time reduction from 3-SAT to APS($\{<, \between\}, \emptyset$).

Lemma 6. *Let I be an instance of the problem* 3-SAT *with n variables and q clauses, and I' an instance $((S,P);(T,Q))$ of* APS($\{<, \between\}, \emptyset$) *obtained by an* APS2-CP*-construction from I. An assignment of the variables that satisfies the boolean formula of I exists iff (T,Q) is an Arc-Preserving Subsequence of (S,P).*

Proof. ($\Rightarrow$) Suppose we have an assignment AS of the n variables that satisfies the boolean formula of I. By definition, for each clause there is at least one literal that satisfies it. Let (S,P) and (T,Q) be two sequences obtained from an APS2-CP-construction from I. We look for a set of bases to delete from S in order to obtain T. We define this set in three steps as follows.

(Step 1) For each variable $x_m \in AS$, $1 \leq m \leq n$:

- if $x_m = True$ then $S^e_{x_m}[2]$ and all the bases of $S^s_{x_m}$ are deleted,
- if $x_m = False$ then $S^e_{x_m}[1]$ and all the bases of $S^s_{\overline{x_m}}$ are deleted.

Notice that the sequence obtained from S_α (resp. S_β) by deleting the bases described above is similar to $T_{\alpha'}$ (resp. $T_{\beta'}$), when not considering arcs.

(Step 2) We recall that, for any $1 \leq m \leq n$ and any $1 \leq i \leq q$, γ^i_m (resp. $\gamma^i_{\overline{m}}$) denotes be the number of occurrences of literal x_m (resp. $\overline{x_m}$) in the set of clauses c_j with $i < j \leq q$ and $\lambda^i_m = \gamma^i_m + \gamma^i_{\overline{m}}$. For any $1 \leq m \leq n$ and for any $1 \leq i \leq q$, we also recall that $y^i_m = 1$ (resp. $y^i_{\overline{m}} = 1$) if $x_m \in c_i$ (resp. $\overline{x_m} \in c_i$), $y^i_m = 0$ (resp. $y^i_{\overline{m}} = 0$) otherwise. For each variable $x_m \in AS$, $1 \leq m \leq n$ and $1 \leq i \leq q$:

- if $x_m = True$ then the following bases are deleted:
  - $rep(i,m,j)[2]$ for all $1 \leq j \leq \lambda^i_m + y^i_{\overline{m}}$,
  - $rep(i,m,j)[1]$ for all $\lambda^i_m + y^i_{\overline{m}} < j \leq 2\lambda^i_m + y^i_{\overline{m}} + y^i_m$,
  - $rep(\overline{i},m,j)[2]$ for all $1 \leq j \leq \lambda^i_m$,
  - $rep(\overline{i},m,j)[1]$ for all $\lambda^i_m < j \leq 2\lambda^i_m$
- if $x_m = False$ then the following bases are deleted:
  - $rep(i,m,j)[1]$ with $1 \leq j \leq \lambda^i_m + y^i_{\overline{m}}$,
  - $rep(i,m,j)[2]$ with $\lambda^i_m + y^i_{\overline{m}} < j \leq 2\lambda^i_m + y^i_{\overline{m}} + y^i_m$,
  - $rep(\overline{i},m,j)[1]$ with $1 \leq j \leq \lambda^i_m$,
  - $rep(\overline{i},m,j)[2]$ with $\lambda^i_m < j \leq 2\lambda^i_m$

Let $j_i \in \{1,2,3\}$ denote the smallest position of the literal(s) satisfying c_i. For each $1 \leq i \leq q$, all the bases of the j_i^{th} terminal of $S^{\overline{i}}$ are deleted.

Notice that, for all $1 \leq m \leq n$ and all $1 \leq i \leq q$, a base G (resp. C) of each repeater of $S^i_{x_m}$ (resp. $S^{\overline{i}}_{x_m}$) is deleted. The sequence obtained from $S^{\overline{i}}$ by deleting the bases described in Step 2 is a sequence of $2 + 2\sum_{m=1}^{n} \lambda^i_m$ substrings

CA (since, by construction, $S^{\overline{i}}$ is initially composed of $2\sum_{m=1}^{n}\lambda_m^i$ repeaters and 3 terminals).

By definition, $\sum_{m=1}^{n}\lambda_m^i$ represents the number of literals in all the clauses c_j with $i < j \leq q$. Since any clause is composed of three literals, we can deduce that $\sum_{m=1}^{n}\lambda_m^i = 3(q-i)$. Therefore, there are $2+2\sum_{m=1}^{n}\lambda_m^i$ (*i.e.* $2+6q-6i$) terminals (*i.e.* CA) in $T^{\overline{i}}$. Consequently, the sequence obtained from $S^{\overline{i}}$ by deleting the bases described in Step 2 is similar to $T^{\overline{i}}$ (when not considering arcs).

(Step 3) For each clause $c_i \in \mathcal{C}_q$ with $1 \leq i \leq q$, the following bases are deleted:

- if exactly one literal (i.e. the j_i^{th}) satisfies c_i then all the bases of the k^{th} and the l^{th} terminals of S^i with $k \neq l$ and $k, l \in \{1,2,3\}\backslash\{j_i\}$.
- if exactly two literals (say the j_i^{th} and k^{th}) satisfy c_i then:
  - all the bases of the l^{th} terminal of S^i with $l \neq k$, $l \neq j_i$ and $l \in \{1,2,3\}$,
  - all the bases of the repeater of S^i connected to the bases of the k^{th} terminal of $S^{\overline{i}}$.
- if exactly three literals (i.e. the j_i^{th}, k^{th} and l^{th}) satisfy c_i then:
  - all the bases of the repeater of S^i connected to the bases of the k^{th} terminal of $S^{\overline{i}}$
  - all the bases of the repeater of S^i connected to the bases of the l^{th} terminal of $S^{\overline{i}}$.

The sequence obtained from S^i by deleting the bases described in Step 2 is composed of a sequence of $6+2\sum_{m=1}^{n}\lambda_m^i$ substrings GA (since, by construction, S^i is initially composed of $3+2\sum_{m=1}^{n}\lambda_m^i$ repeaters and 3 terminals). Moreover, we know that $\sum_{m=1}^{n}\lambda_m^i = 3(q-i)$. Therefore, there are $4+2\sum_{m=1}^{n}\lambda_m^i$ (*i.e.* $4+6q-6i$) terminals (*i.e.* substrings GA) in T^i. As in each of the above cases, all the bases of two elements of S^i have been deleted, the sequence obtained from S^i by deleting the bases described in Step 2 and Step 3 is similar to T^i (when not considering arcs).

Thus, the sequence obtained from S by deleting the bases described in Step 1, Step 2 and Step 3 is similar to T (when not considering arcs). We now turn to demonstrating that at least one base of any arc of P has been deleted. In the following, we will distinguish arcs between bases A and U, denoted by AU-arcs, from arcs between bases C and G, denoted by CG-arcs. Let us consider those two types of arcs separately:

(1) By construction, for all $1 \leq m \leq n$, the following AU-arcs have been created: $(S^e_{x_m}[1], S^s_{x_m}[1])$ and $(S^e_{x_m}[2], S^s_{\overline{x}_m}[k_m+1])$.
By Step 1, since a variable x_m has a unique value, either each base of $S^s_{\overline{x}_m}$ and $S^e_{x_m}[1]$, or each base of $S^s_{x_m}$ and $S^e_{x_m}[2]$ is deleted for all $1 \leq m \leq n$. Thus, at least one base in S of any AU-arc of P is deleted.

(2) By construction, the following CG-arcs have been created:
- for all $1 \leq m \leq n$, $1 \leq j \leq 2\lambda_m^i$ and $1 \leq i < q$:
  - an arc between the second base G of $rep(i,m,j)$ and the first base C of the j^{th} element (i.e. either a terminal or a repeater) of $S^{\overline{i}}_{x_m}$;

- an arc between the second base C of $rep(\overline{i}, m, j)$ and the first base G of the j^{th} element of $S^{i+1}_{x_m}$.

– for all $1 \leq j \leq \gamma_m + \gamma_{\overline{m}}$, an arc between the j^{th} base C of substring $S^s_{x_m} A S^s_{\overline{x_m}}$ in S_β and the first base G of the j^{th} element of $S^1_{x_m}$ in S_ζ.

In the following, we focus on the arcs of a clause c_i and the arcs between c_i and c_{i+1}, for any given $1 \leq i < q$ (cf. Figure 6). More precisely, we will demonstrate that, for any given $1 \leq m \leq n$, at least one base of any arc in $\{S^i_m, S^{\overline{i}}_m, S^{i+1}_m, S^{\overline{i+1}}_m\}$ is deleted. This will prove that at least one base of any arc connecting two bases of S_ζ is deleted. In a second step, we will focus on the first clause and prove that at least one base of any arc connecting a base of S_β and a base of S^1 is deleted.

We recall that by construction:

$$S^i_{x_m} = (GGA)^{\lambda^i_m + y^i_{\overline{m}}} (GA)^{y^i_m} (GGA)^{\lambda^i_m + y^i_m} (GA)^{y^i_{\overline{m}}}$$
$$S^{\overline{i}}_{x_m} = (CCA)^{\lambda^i_m} (CA)^{y^i_{\overline{m}}} (CCA)^{\lambda^i_m} (CA)^{y^i_m}$$

Consider any variable x_m with $1 \leq m \leq n$. For any given $1 \leq m \leq n$ and $1 \leq i \leq q$, we define the following four subsets of arcs:

– (A^i_m) for each $1 \leq m \leq n$, the $\lambda^i_m + y^i_{\overline{m}}$ first arcs between a base of $S^i_{x_m}$ and a base of $S^{\overline{i}}_{x_m}$;
– (B^i_m) for each $1 \leq m \leq n$, the rest of the arcs between a base of $S^i_{x_m}$ and a base of $S^{\overline{i}}_{x_m}$;
– (C^i_m) for each $1 \leq m \leq n$, the λ^i_m first arcs between a base of $S^{\overline{i}}_{x_m}$ and a base of $S^{i+1}_{x_m}$;
– (D^i_m) for each $1 \leq m \leq n$, the rest of the arcs between a base of $S^{\overline{i}}_{x_m}$ and a base of $S^{i+1}_{x_m}$.

Suppose first that $x_m = True$. We now consider separately the nine following cases:

– (a_1) $x_m, x_{\overline{m}} \notin \{c_i, c_{i+1}\}$;
– (a_2) $x_m, x_{\overline{m}} \notin c_i$ and $x_m \in c_{i+1}$;
– (a_3) $x_m, x_{\overline{m}} \notin c_i$ and $x_{\overline{m}} \in c_{i+1}$;
– (b_1) $x_m \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$;
– (b_2) $x_m \in c_i$ and $x_m \in c_{i+1}$;
– (b_3) $x_m \in c_i$ and $x_{\overline{m}} \in c_{i+1}$;
– (c_1) $x_{\overline{m}} \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$;
– (c_2) $x_{\overline{m}} \in c_i$ and $x_m \in c_{i+1}$;
– (c_3) $x_{\overline{m}} \in c_i$ and $x_{\overline{m}} \in c_{i+1}$.

(a_1). Since $x_m, x_{\overline{m}} \notin \{c_i, c_{i+1}\}$, by definition, $y^i_m = y^i_{\overline{m}} = y^{i+1}_m = y^{i+1}_{\overline{m}} = 0$. Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda^i_m + y^i_{\overline{m}}$, $rep(i, m, j)[2]$ is deleted. Thus, at least one base of any arc of the set (A^i_m) is deleted.

Since $x_m = True$, $rep(\overline{i}, m, j)[1]$ is deleted for all $1 \leq i \leq q$ and all $\lambda^i_m < j \leq 2\lambda^i_m$ (cf. Step 2). Therefore, at least one base of any arc of the set (B^i_m) is deleted.

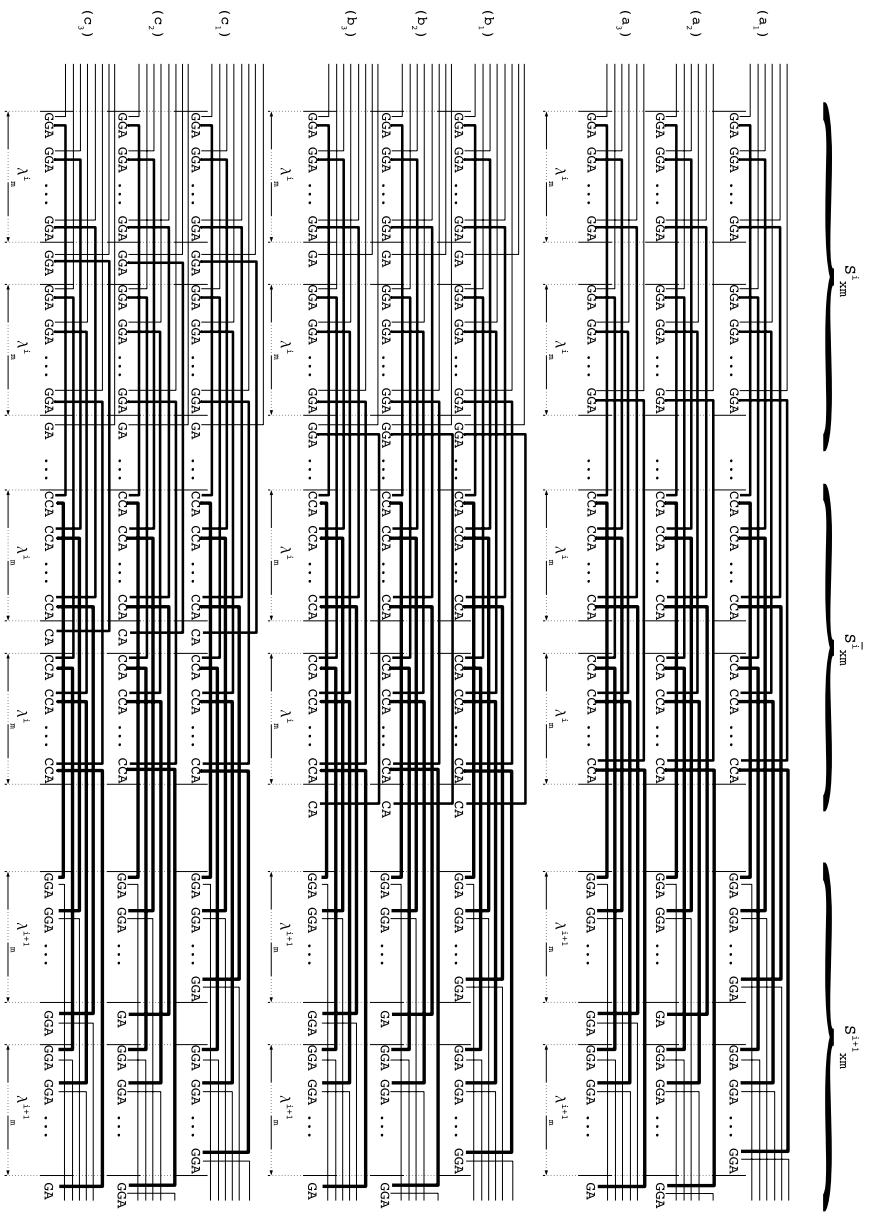


Fig. 6. Sketch of the arc-structure of a clause c_i, for any given $1 \leq m \leq n$ and $1 \leq i < q$. (a_1) when $x_m, x_{\overline{m}} \notin \{c_i, c_{i+1}\}$. (a_2) when $x_m, x_{\overline{m}} \notin c_i$ and $x_m \in c_{i+1}$. (a_3) when $x_m, x_{\overline{m}} \notin c_i$ and $x_{\overline{m}} \in c_{i+1}$. (b_1) when $x_m \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$. (b_2) when $x_m \in c_i$ and $x_m \in c_{i+1}$. (b_3) when $x_m \in c_i$ and $x_{\overline{m}} \in c_{i+1}$. (c_1) when $x_{\overline{m}} \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$. (c_2) when $x_{\overline{m}} \in c_i$ and $x_m \in c_{i+1}$. (c_3) when $x_{\overline{m}} \in c_i$ and $x_{\overline{m}} \in c_{i+1}$.

Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i$, $rep(\overline{i}, m, j)[2]$ is deleted (cf. Step 2). Consequently, at least one base of any arc of the set (C_m^i) is deleted.

Finally, $x_m = True$ implies that $rep(i+1, m, j)[1]$ is deleted for all $1 \leq i < q$ and all $\lambda_m^{i+1} + y_{\overline{m}}^{i+1} < j \leq 2\lambda_m^{i+1} + y_{\overline{m}}^{i+1} + y_m^{i+1}$. Therefore, at least one base of any arc of the set (D_m^i) is deleted.

(a_2). The proof is fully similar to the one of (a_1).

(a_3). Since $x_m, x_{\overline{m}} \notin c_i$ and $x_{\overline{m}} \in c_{i+1}$, by definition, $y_m^i = y_{\overline{m}}^i = y_m^{i+1} = 0$ and $y_{\overline{m}}^{i+1} = 1$. Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i + y_{\overline{m}}^i$, $rep(i, m, j)[2]$ is deleted. Thus, at least one base of any arc of the set (A_m^i) is deleted.

Since $x_m = True$, $rep(\overline{i}, m, j)[1]$ is deleted for all $1 \leq i \leq q$ and all $\lambda_m^i < j \leq 2\lambda_m^i$ (cf. Step 2). Therefore, at least one base of any arc of the set (B_m^i) is deleted.

Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i$, $rep(\overline{i}, m, j)[2]$ is deleted (cf. Step 2). Consequently, at least one base of any arc of the set (C_m^i) is deleted.

Finally, $x_m = True$ implies that $rep(i+1, m, j)[1]$ is deleted for all $1 \leq i < q$ and all $\lambda_m^{i+1} + y_{\overline{m}}^{i+1} < j \leq 2\lambda_m^{i+1} + y_{\overline{m}}^{i+1} + y_m^{i+1}$. Moreover, by construction, if $y_{\overline{m}}^{i+1} = 1$ then there is an arc connecting the base $rep(\overline{i}, m, j)[2]$ to a base of the j^{th} element (which is a terminal) of $S_{x_m}^{i+1}$ where $j = 2\lambda_m^i$. By definition, as $\overline{x_m} \in c_{i+1}$, $\overline{x_m}$ does not satisfies c_{i+1} (since $x_m = True$). By definition, there exists at least a literal which, by it assignment, satisfies c_{i+1}. Therefore, all the bases of the terminal of $S_{\overline{x_m}}^{i+1}$ have been deleted (cf. Step 3). Therefore, at least one base of any arc of the set (D_m^i) is deleted.

(b_1). Since $x_m \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$, by definition, $y_m^i = 1$ and $y_{\overline{m}}^i = y_m^{i+1} = y_{\overline{m}}^{i+1} = 0$. Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i + y_{\overline{m}}^i$, $rep(i, m, j)[2]$ is deleted. Thus, at least one base of any arc of the set (A_m^i) is deleted.

Since $x_m = True$, $rep(\overline{i}, m, j)[1]$ is deleted for all $1 \leq i \leq q$ and all $\lambda_m^i < j \leq 2\lambda_m^i$ (cf. Step 2). Moreover, by construction, if $y_m^i = 1$ then there is an arc connecting the base $rep(i, m, j)[2]$ to a base of the j^{th} element (which is a terminal) of $S_{x_m}^{\overline{i}}$ where $j = 2\lambda_m^i + y_{\overline{m}}^i + y_m^i$. By definition, since $y_m^i = 1$, $x_m \in c_i$ and thus x_m satisfies c_i. If x_m is the literal with the smallest position of the literal(s) satisfying c_i, then all the bases of the terminal of $S_{x_m}^{\overline{i}}$ have been deleted. Otherwise, all the bases of the repeater of $S_{x_m}^i$ connected to the bases of the terminal of $S_{x_m}^{\overline{i}}$ are deleted (cf. Step 3). Therefore, at least one base of any arc of the set (B_m^i) is deleted.

Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i$, $rep(\overline{i}, m, j)[2]$ is deleted (cf. Step 2). Consequently, at least one base of any arc of the set (C_m^i) is deleted.

Finally, $x_m = True$ implies that $rep(i+1,m,j)[1]$ is deleted for all $1 \leq i < q$ and all $\lambda_m^{i+1} + y_{\overline{m}}^{i+1} < j \leq 2\lambda_m^{i+1} + y_{\overline{m}}^{i+1} + y_m^{i+1}$. Therefore, at least one base of any arc of the set (D_m^i) is deleted.

(b_2). The proof is fully similar to the one of (b_1).

(b_3). Since $x_m \in c_i$ and $x_{\overline{m}} \in c_{i+1}$, by definition, $y_m^i = y_{\overline{m}}^{i+1} = 1$ and $y_m^{i+1} = y_{\overline{m}}^i = 0$. Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i + y_{\overline{m}}^i$, $rep(i,m,j)[2]$ is deleted. Thus, at least one base of any arc of the set (A_m^i) is deleted.

Since $x_m = True$, $rep(\overline{i},m,j)[1]$ is deleted for all $1 \leq i \leq q$ and all $\lambda_m^i < j \leq 2\lambda_m^i$ (cf. Step 2). Moreover, by construction, if $y_m^i = 1$ then there is an arc connecting the base $rep(i,m,j)[2]$ to a base of the j^{th} element (which is a terminal) of $S_{x_m}^{\overline{i}}$ where $j = 2\lambda_m^i + y_{\overline{m}}^i + y_m^i$. By definition, since $y_m^i = 1$, $x_m \in c_i$ and thus x_m satisfies c_i. If x_m is the literal with the smallest position of the literal(s) satisfying c_i then all the bases of the terminal of $S_{x_m}^{\overline{i}}$ have been deleted. Otherwise, all the bases of the repeater of $S_{x_m}^i$ connected to the bases of the terminal of $S_{x_m}^{\overline{i}}$ are deleted (cf. Step 3). Therefore, at least a base of any arc of the set (B_m^i) is deleted.

Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i$, $rep(\overline{i},m,j)[2]$ is deleted (cf. Step 2). Consequently, at least one base of any arc of the set (C_m^i) is deleted.

Finally, $x_m = True$ implies that $rep(i+1,m,j)[1]$ is deleted for all $1 \leq i < q$ and all $\lambda_m^{i+1} + y_{\overline{m}}^{i+1} < j \leq 2\lambda_m^{i+1} + y_{\overline{m}}^{i+1} + y_m^{i+1}$. Moreover, by construction, if $y_{\overline{m}}^{i+1} = 1$ then there is an arc connecting the base $rep(\overline{i},m,j)[2]$ to a base of the j^{th} element (which is a terminal) of $S_{x_m}^{i+1}$ where $j = 2\lambda_m^i$. By definition, as $\overline{x_m} \in c_{i+1}$, $\overline{x_m}$ does not satisfies c_{i+1} (since $x_m = True$). By definition, there exists at least a literal which, by it assignment, satisfies c_{i+1}. Therefore, all the bases of the terminal of $S_{\overline{x_m}}^{i+1}$ have been deleted (cf. Step 3). Therefore, at least one base of any arc of the set (D_m^i) is deleted.

(c_1). Since $x_{\overline{m}} \in c_i$ and $x_m, x_{\overline{m}} \notin c_{i+1}$, by definition, $y_{\overline{m}}^i = 1$ and $y_m^i = y_m^{i+1} = y_{\overline{m}}^{i+1} = 0$. Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i + y_{\overline{m}}^i$, $rep(i,m,j)[2]$ is deleted. Thus, at least one base of any arc of the set (A_m^i) is deleted.

Since $x_m = True$, $rep(\overline{i},m,j)[1]$ is deleted for all $1 \leq i \leq q$ and all $\lambda_m^i < j \leq 2\lambda_m^i$ (cf. Step 2). Therefore, at least a base of any arc of the set (B_m^i) is deleted.

Moreover, as $x_m = True$, for all $1 \leq i \leq q$ and all $1 \leq j \leq \lambda_m^i$, $rep(\overline{i},m,j)[2]$ is deleted (cf. Step 2). Consequently, at least one base of any arc of the set (C_m^i) is deleted.

Finally, $x_m = True$ implies that $rep(i+1,m,j)[1]$ is deleted for all $1 \leq i < q$ and all $\lambda_m^{i+1} + y_{\overline{m}}^{i+1} < j \leq 2\lambda_m^{i+1} + y_{\overline{m}}^{i+1} + y_m^{i+1}$. Moreover, by construction, if $y_{\overline{m}}^{i+1} = 1$ then there is an arc connecting the base $rep(\overline{i},m,j)[2]$ to a base of the j^{th} element (which is a terminal) of $S_{x_m}^{i+1}$ where $j = 2\lambda_m^i$. By definition, as $\overline{x_m} \in c_{i+1}$, $\overline{x_m}$ does not satisfies c_{i+1} (since $x_m = True$). By definition, there

exists at least a literal which, by it assignment, satisfies c_{i+1}. Therefore, all the bases of the terminal of $S^{i+1}_{\overline{x_m}}$ have been deleted (cf. Step 3). Therefore, at least one base of any arc of the set (D^i_m) is deleted.

(c_2). The proof is fully similar to the one of (c_1).

(c_3). The proof is fully similar to the one of (a_1).

Therefore, when $x_m = True$, at least one base of any CG-arc has been deleted. If $x_m = False$ then a similar reasoning leads to the same conclusion, *i.e.* at least one base of any CG-arc has been deleted. Thus, for any $1 < i \leq q$, any CG-arc between a base of an element of the representation of the clause c_{i-1} (*i.e.* $S^{i-1}\ U\ S^{\overline{i-1}}$) and a base of an element of the representation of the clause c_i (*i.e.* $S^i\ U\ S^{\overline{i}}$) has been deleted.

Moreover, for any $1 \leq i \leq q$, any CG-arc between two bases of the representation of the clause c_i has been deleted. Remains us to consider the special case of the first clause (*i.e.* c_1). Indeed, there is, for all $1 \leq j \leq \gamma_m + \gamma_{\overline{m}}$, an arc between the j^{th} base C of substring $S^s_{x_m} A S^s_{\overline{x_m}}$ in S_β and the first base G of the j^{th} element of $S^1_{x_m}$ in S_ζ.

For each $1 \leq m \leq n$, if $x_m = True$ then each base of $S^s_{x_m}$ and $S^e_{x_m}[2]$ is deleted and $rep(1, m, j)[2]$ is deleted with $1 \leq j \leq \lambda^i_m + y^i_{\overline{m}}$. Moreover, for each $1 \leq m \leq n$, if $x_m = False$ then each base of $S^s_{\overline{x_m}}$ and $S^e_{x_m}[1]$ is deleted and $rep(1, m, j)[1]$ is deleted with $1 \leq j \leq \lambda^i_m + y^i_{\overline{m}}$. Thus, at least one base in S of any CG-arc of P is deleted.

We just proved that if S' is the sequence obtained from S by deleting all the bases described in Step 1, Step 2 and Step 3 together with their incident arcs, then there is no arc in S' (i.e. neither AU-arcs or CG-arcs). Moreover, we demonstrated previously that the sequence S' is similar to T. Therefore, if an assignment of the variables that satisfies the boolean formula of I exists, then (T, Q) is an Arc-Preserving Subsequence of (S, P).

($\Leftarrow$) Let I be an instance of the problem 3-SAT with n variables and q clauses. Let I' be an instance $((S, P); (T, Q))$ of APS($\{<, \between\}, \emptyset$) obtained by an APS2-CP-construction from I such that (T, Q) can be obtained from (S, P) by deleting some of its bases together with their incident arcs, if any. By Lemma 3, any corresponding alignment of (S, P) and (T, Q) is canonical. Therefore, $T^s_{x_m}$ is matched with either $S^s_{x_m} A$ or $A\ S^s_{\overline{x_m}}$. Consequently, for any $1 \leq m \leq n$, we define an assignment AS of the variables of I as follows:

- if $T^s_{x_m}$ is matched with $S^s_{x_m} A$ then $x_m = False$,
- otherwise, $x_m = True$.

Now, let us prove that for any $1 \leq i \leq q$ the clause c_i is satisfied by AS. Let us first focus on the first clause (*i.e.* c_1). c_1 is defined by three literals (say x_i, x_j and x_k). Since, c_1 is equal to the disjunction of variables built with x_i, x_j and x_k, c_1 can have eight different forms, because each literal can appear in either

its positive (x_i) or negative ($\overline{x_i}$) form. In the following, we suppose, to illustrate the proof, that $c_1 = (x_i \vee x_j \vee \overline{x_k})$ as illustrated in Figure 5, since the other cases can be treated similarly.

By Lemmas 4 and 5, the two following properties must be satisfied:

- all the repeaters and two terminals of $S^{\overline{1}}$ are active,
- and either:
  - all the repeaters and one terminal of S^1 are active,
  - all the repeaters but one and two terminals of S^1 are active,
  - all the repeaters but two and three terminals of S^1 are active.

(1) Suppose that all the repeaters of S^1 and one terminal of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. The active terminal can be in either $S^1_{x_i}$, $S^1_{x_j}$ or $S^1_{x_k}$. Since the cases where the active terminal is either in $S^1_{x_i}$ or $S^1_{x_j}$ are fully similar, we detail hereafter only two cases: (a) the active terminal is in $S^1_{x_i}$ and (b) the active terminal is in $S^1_{x_k}$.

(a) Suppose that the active terminal is in $S^1_{x_i}$. By construction, there is an arc between a base C of $S^s_{x_i}$ and the first base of the terminal in $S^1_{x_i}$. Thus, a base C of $S^s_{x_i}$ is deleted. Therefore, by the way we defined AS, $x_i = True$ and thus c_1 is satisfied.

(b) Suppose that the active terminal is in $S^1_{x_k}$. By construction, there is an arc between a base C of $S^s_{\overline{x_k}}$ and the first base of the terminal in $S^1_{x_k}$. Thus, a base C of $S^s_{\overline{x_k}}$ is deleted. Therefore, by the way we defined AS, $x_k = False$ and thus c_1 is satisfied.

(2) Suppose that all the repeaters but one of S^1 and two terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. The active terminals can be in either $(S^1_{x_i}, S^1_{x_j})$, $(S^1_{x_i}, S^1_{x_k})$ or $(S^1_{x_j}, S^1_{x_k})$. Since the cases where the active terminals are either in $(S^1_{x_i}, S^1_{x_k})$ or $(S^1_{x_j}, S^1_{x_k})$ are fully similar, we detail hereafter only two cases: (a) the active terminals are in $(S^1_{x_i}, S^1_{x_j})$ and (b) the active terminals are in $(S^1_{x_i}, S^1_{x_k})$.

(a) Suppose that the active terminals are in $(S^1_{x_i}, S^1_{x_j})$. By construction, there is an arc between a base C of $S^s_{x_i}$ and the first base of the terminal in $S^1_{x_i}$. Thus, a base C of $S^s_{x_i}$ is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{x_j}$ and the first base of the terminal in $S^1_{x_j}$. Thus, a base C of $S^s_{x_j}$ is deleted. Therefore, by the way we defined AS, $x_i = x_j = True$ and thus c_1 is satisfied.

For the sake of the proof, we now detail the alignment of the elements of c_1 in case (a). Since all the repeaters and two terminals of $S^{\overline{1}}$ are active, at least a terminal of either $S^{\overline{1}}_{x_i}$ or $S^{\overline{1}}_{x_j}$ is active. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$ and $(rep[2], \theta) \in P$, where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Moreover, by construction, there is a repeater rep' of $S^1_{x_j}$ such that $(\delta', rep'[1]) \in P$ and $(rep'[2], \theta') \in P$, where δ' (resp. θ') is a base C of $S^s_{\overline{x_j}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_j}$), as illustrated in Figure 5. Since at

least a terminal of either $S^{\overline{1}}_{x_i}$ or $S^{\overline{1}}_{x_j}$ is active, then either θ or θ' is matched. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. Therefore, either $rep[2]$ or $rep'[2]$ is deleted. Since either rep or rep' is an active repeater, either $rep[1]$ or $rep'[1]$ is matched. Thus, either δ or δ' is deleted. Since the alignment is canonical, for all $1 \leq m \leq n$, a base of both $S^s_{x_m}$ and $S^s_{\overline{x_m}}$ cannot be deleted. Therefore, the only two solutions are: either the terminal of $S^{\overline{1}}_{x_i}$ and rep' are inactive, or the terminal of $S^{\overline{1}}_{x_j}$ and rep are inactive.

(b) Suppose that the active terminals are in $(S^1_{x_i}, S^1_{x_k})$. By construction, there is an arc between a base C of $S^s_{x_i}$ and the first base of the terminal in $S^1_{x_i}$. Thus, a base C of $S^s_{x_i}$ is deleted. Moreover, by construction, there is an arc between a base C of $S^s_{\overline{x_k}}$ and the first base of the terminal in $S^1_{x_k}$. Thus, a base C of $S^s_{\overline{x_k}}$ is deleted. Therefore, by the way we defined AS, $x_i = True$, $x_k = False$ and thus c_1 is satisfied.

For the sake of the proof, we now detail the alignment of the elements of c_1 in case (b). Since all the repeaters and two terminals of $S^{\overline{1}}$ are active, at least a terminal of either $S^{\overline{1}}_{x_i}$ or $S^{\overline{1}}_{x_k}$ is active. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$ and $(rep[2], \theta) \in P$, where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Moreover, by construction, there is a repeater rep' of $S^1_{x_k}$ such that $(\delta', rep'[1]) \in P$ and $(rep'[2], \theta') \in P$, where δ' (resp. θ') is a base C of $S^s_{x_k}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_k}$), as illustrated in Figure 5. Since at least a terminal of either $S^{\overline{1}}_{x_i}$ or $S^{\overline{1}}_{x_k}$ is active, then either θ or θ' is matched. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. Therefore, either $rep[2]$ or $rep'[2]$ is deleted. Since either rep or rep' is an active repeater, either $rep[1]$ or $rep'[1]$ is matched. Thus, either δ or δ' is deleted. Since the alignment is canonical, for all $1 \leq m \leq n$, a base of both $S^s_{x_m}$ and $S^s_{\overline{x_m}}$ cannot be deleted. Therefore, the only two solutions are: either the terminal of $S^{\overline{1}}_{x_i}$ and rep' are inactive, or the terminal of $S^{\overline{1}}_{x_k}$ and rep are inactive.

(3) Suppose that all the repeaters but two of S^1 and three terminals of $\{S^1_{x_i}, S^1_{x_j}, S^1_{x_k}\}$ are active. By construction, there is an arc between a base C of $S^s_{x_i}$ and the first base of the terminal in $S^1_{x_i}$. Thus, a base C of $S^s_{x_i}$ is deleted. Moreover, there is an arc between a base C of $S^s_{x_j}$ and the first base of the terminal in $S^1_{x_j}$. Thus, a base C of $S^s_{x_j}$ is deleted. Finally, by construction, there is an arc between a base C of $S^s_{\overline{x_k}}$ and the first base of the terminal in $S^1_{x_k}$. Thus, a base C of $S^s_{\overline{x_k}}$ is deleted. Therefore, by the way we defined AS, $x_i = x_j = True$, $x_k = False$ and thus c_1 is satisfied.

For the sake of the proof, we now detail the alignment of the elements of c_1 in case (3). Since all the repeaters and two terminals of $S^{\overline{1}}$ are active, at least two terminals of $S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}$ are active. By construction, there is a repeater rep of $S^1_{x_i}$ such that $(\delta, rep[1]) \in P$ and $(rep[2], \theta) \in P$, where δ (resp. θ) is a base C of $S^s_{\overline{x_i}}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_i}$), as illustrated in Figure 5. Moreover, by construction, there is a repeater rep' of $S^1_{x_j}$ such that $(\delta', rep'[1]) \in P$ and $(rep'[2], \theta') \in P$, where δ' (resp. θ') is a base C of $S^s_{\overline{x_j}}$ (resp. the first base

of the terminal in $S^{\overline{1}}_{x_j}$), as illustrated in Figure 5. Finally, by construction, there is a repeater rep'' of $S^1_{x_k}$ such that $(\delta'', rep''[1]) \in P$ and $(rep''[2], \theta'') \in P$, where δ'' (resp. θ'') is a base C of $S^s_{x_k}$ (resp. the first base of the terminal in $S^{\overline{1}}_{x_k}$), as illustrated in Figure 5. Since at least two terminals of $S^{\overline{1}}_{x_i}, S^{\overline{1}}_{x_j}, S^{\overline{1}}_{x_k}$ are active, then at least two of $(\theta, \theta', \theta'')$ are matched. By definition, as $Q = \emptyset$, at least one base incident to every arc of P has to be deleted. Therefore, two of $(rep[2], rep'[2], rep''[2])$ are deleted. Since rep, rep' or rep'' is an active repeater, either $rep[1]$, $rep'[1]$ or $rep''[1]$ is matched. Thus, either δ, δ' or δ'' is deleted. Since the alignment is canonical, for all $1 \leq m \leq n$ a base of both $S^s_{x_m}$ and $S^s_{\overline{x_m}}$ cannot be deleted. Therefore, the only three solutions are: either the terminal of $S^{\overline{1}}_{x_i}$ and rep' and rep'' are inactive, or the terminal of $S^{\overline{1}}_{x_j}$ and rep and rep'' are inactive, or the terminal of $S^{\overline{1}}_{x_k}$ and rep and rep' are inactive.

We just proved that if I' is a solution then the truth assignment we defined above satisfies clause c_1. Moreover, we proved that any inactive repeater of S^1 is linked to a terminal of $S^{\overline{1}}$ (*i.e.* its second base is connected to a base of a terminal of $S^{\overline{1}}$). Let rep be a repeater in S such that $rep[1]$ and $rep[2]$ are respectively connected to bases u and v. The particular design of the repeaters ensues that if rep is active then the situation is equivalent to the one where u and v are connected with an arc. Indeed, if (S, P) is an arc-preserving subsequence of (T, Q) and rep is active, then exactly one out of $\{rep[1], rep[2]\}$ is matched. Therefore, if v is matched then $rep[2]$ is deleted and $rep[1]$ is matched. Consequently, u is deleted. Similarly, if u is matched then v is deleted. More generally we can prove the following claim (illustrated in Figure 7):

Claim. Let u and v be two bases and $\{rep_1, rep_2 \dots rep_k\}$ be a set of repeaters such that $(u, rep_1[1]) \in P$, $(rep_k[2], v) \in P$ and $(rep_i[2], rep_{i+1}[1]) \in P$ for all $1 \leq i < k$.

Let $\mathcal{A}$ be an alignment. If for each $1 \leq i \leq k$, rep_i is active in $\mathcal{A}$, then:

- if u is matched then v is deleted;
- if v is matched then u is deleted.

Therefore, since all the repeaters of $S^{\overline{1}}$ are active and the inactive repeaters of S^1 are linked to terminals of $S^{\overline{1}}$, by the above claim, considering clause c_2 is equivalent to considering c_1. Therefore, c_2 is satisfied and all the repeaters of $S^{\overline{2}}$ are active and the inactive repeaters of S^2 are linked to terminals of $S^{\overline{2}}$. Consequently, a similar reasoning can be done recursively for any clause c_i with $1 \leq i \leq q$. Thus, we just proved that if I' is a solution then the truth assignment we defined above satisfies all the clauses. □

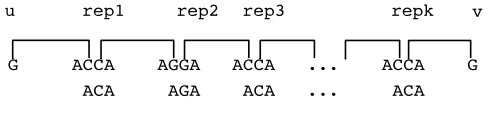


Fig. 7. Illustration of Claim 4

5 Two Polynomial Time Solvable APS Problems

We prove in this section that APS($\{\between\}$,$\emptyset$) and APS($\{\between\}$,$\{\between\}$) are polynomial time solvable. In other words, the relation $\between$ alone does not imply NP-completeness.

We need the following notations. Sequences are the concatenation of zero or more elements from an alphabet. We use the period "." as the concatenation operator, but frequently the two operands are simply put side by side. Let $S =$

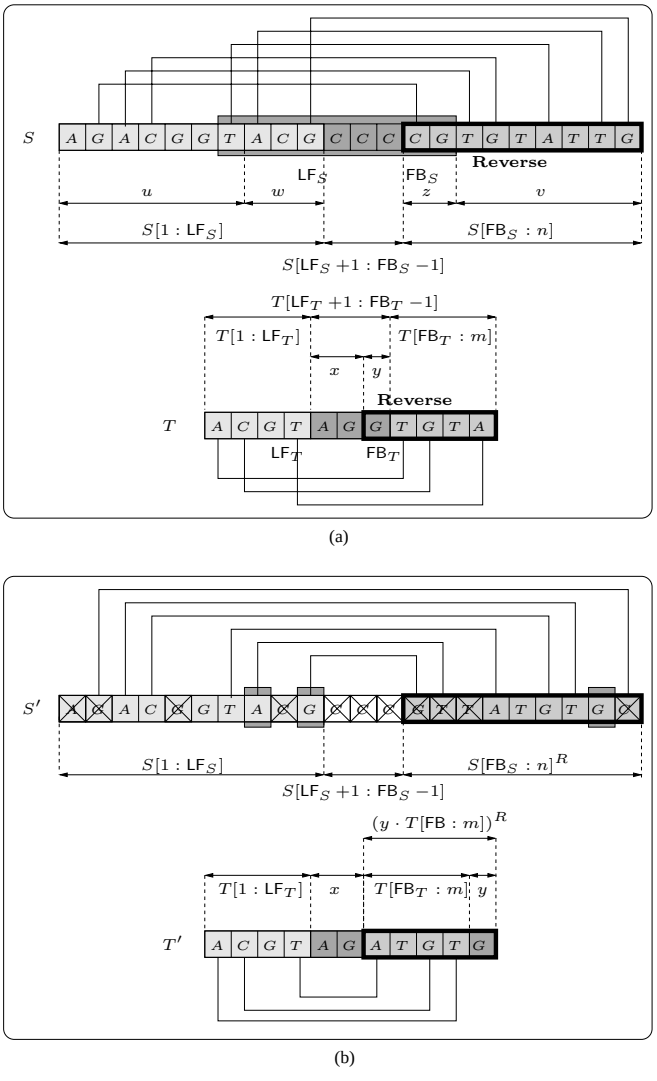


Fig. 8. Illustration of Lemma 7

$S[1]\,S[2]\ldots S[m]$ be a sequence of length m. For all $1 \leq i \leq j \leq m$, we write $S[i:j]$ to denote $S[i]\,S[i+1]\ldots S[j]$. The *reverse* of S is the sequence $S^R = S[m]\ldots S[2]\,S[1]$. A *factorization* of S is any decomposition $S = x_1\,x_2\ldots x_q$ where $x_1, x_2, \ldots x_q$ are (possibly empty) sequences. Let (S,P) be a $\{\langle\rangle\}$-arc-annotated sequence and $(i,j) \in P$, $i < j$, be an arc. We call $S[i]$ a *forward base* and $S[j]$ a *backward base*. We will denote by LF_S the position of the last forward base in (S,P) and by FB_S the position of the first backward base in (S,P), *i.e.*, $\mathsf{LF}_S = \max\{i : (i,j) \in P\}$ and $\mathsf{FB}_S = \min\{j : (i,j) \in P\}$. By convention, we let $\mathsf{LF}_S = 0$ and $\mathsf{FB}_S = |S| + 1$ if $P = \emptyset$. Observe that $\mathsf{LF}_S < \mathsf{FB}_S$.

We begin by proving a factorization result on $\{\langle\rangle\}$-arc-annotated sequences.

Lemma 7. *Let S and T be two $\{\langle\rangle\}$-arc-annotated sequences of length n and m, respectively. If T occurs as an arc preserving subsequence in S, then there exists a possibly trivial factorization $T[\mathsf{LF}_T+1 : \mathsf{FB}_T-1] = xy$ such that $T[1 : \mathsf{LF}_T] \cdot x \cdot (y \cdot T[\mathsf{FB}_T : m])^R$ occurs as an arc preserving subsequence in $S[1 : \mathsf{FB}_S-1] \cdot S[\mathsf{FB}_S : n]^R$.*

Proof. Suppose that T occurs as an arc preserving subsequence in S. Since both S and T are $\{\langle\rangle\}$-arc-annotated sequences, then there exist two factorizations $S[1 : \mathsf{LF}_S] = uw$ and $S[\mathsf{FB}_S : n] = zv$ such that: (i) $T[1 : \mathsf{LF}_T]$ occurs in u, (ii) $T[\mathsf{LF}_T+1 : \mathsf{FB}_T-1]$ occurs in $w \cdot S[\mathsf{LF}_S+1 : \mathsf{FB}_S-1] \cdot z$ and (iii) $T[\mathsf{FB}_T : m]$ occurs in v. Then it follows that there exists a factorization $T[\mathsf{LF}_T+1 : \mathsf{FB}_T-1] = xy$ such that x occurs in $w \cdot S[\mathsf{LF}_S+1 : \mathsf{FB}_S-1]$ and y occurs in z, and hence $T' = T[1 : \mathsf{LF}_T] \cdot x \cdot (y \cdot T[\mathsf{FB}_T : m])^R$ occurs as an arc preserving subsequence in $S' = S[1 : \mathsf{FB}_S-1] \cdot S[\mathsf{FB}_S : n]^R$ (see Figure 8). □

Theorem 3. *The* APS($\{\langle\rangle\}$,$\{\langle\rangle\}$) *problem is solvable in $O(nm^2)$ time.*

Proof. The algorithm we propose is Algorithm 1.

Algorithm 1: An $O(nm^2)$ time algorithm solving the APS($\{\langle\rangle\}$,$\{\langle\rangle\}$) problem

```
Data   : Two {⟨⟩}-arc-annotated sequences S and T of length n and m, respectively
Result : true iff T occurs as an arc-preserving subsequence in S
begin
1   S' = S[1 : FB_S − 1] · S[FB_S : n]^R
2   foreach factorization T[LF_T + 1 : FB_T − 1]| = xy do
3       T' = T[1 : LF_T] · x · (y · T[FB_T : m])^R
4       if T' occurs as an arc preserving subsequence in S' then
5           return true
6   return false
end
```

Correctness of the algorithm follows from Lemma 7. What is left is to prove the time complexity. Clearly, $S' = S[1 : \mathsf{FB}_S-1] \cdot S[\mathsf{FB}_S : n]^R$ is a $\{\sqsubset\}$-arc-annotated sequence. The key point is to note that, for any factorization

$T[\mathsf{LF}_T+1 : \mathsf{FB}_T-1] = xy$, the obtained $T' = T[1 : \mathsf{LF}_T] \cdot x \cdot (y \cdot T[\mathsf{FB}_T : m])^R$ is a $\{\sqsubset\}$-arc-annotated sequence as well. Now let k be the number of arcs in T. So there are at most $m - 2k$ iterations to go before eventually returning false. According to the above, Line 4 constitutes an instance of APS($\{\sqsubset\}$,$\{\sqsubset\}$). But APS($\{\sqsubset\}$,$\{\sqsubset\}$) is a special case of APS($\{<,\sqsubset\}$,$\{<,\sqsubset\}$), and hence is solvable in $O(nm)$ time [11]. Then it follows that the algorithm as a whole runs in $O(nm(m-2k)) = O(nm^2)$ time. □

Clearly, proof of Theorem 3 relies on an efficient algorithm for solving APS ($\{\sqsubset\}$,$\{\sqsubset\}$): the better the complexity for APS($\{\sqsubset\}$,$\{\sqsubset\}$), the better the complexity for APS($\{\between\}$,$\{\between\}$). We have used only the fact that APS($\{\sqsubset\}$,$\{\sqsubset\}$) is a special case of APS($\{<,\sqsubset\}$,$\{<,\sqsubset\}$). It remains open, however, wether a better complexity can be achieved for APS($\{\sqsubset\}$,$\{\sqsubset\}$).

Theorem 3 carries out easily to restricted versions (Observation 1).

Corollary 1. APS($\{\between\}$,$\emptyset$) *is solvable in* $O(nm^2)$ *time.*

6 Conclusion

In this paper, we investigated the APS problem time complexity and gave a precise characterization of what makes the APS problem hard. We proved that APS(Crossing,Plain) is **NP**-complete thereby answering an open problem posed in [11] (see Table 3). Note that this result answers the last open problem concerning APS computational complexity with respect to classical complexity levels, *i.e.*, Plain, Chain, Nested and Crossing. Also, we refined the four above mentioned levels for exploring the border between polynomial time solvable and **NP**-complete problems. We proved that both APS($\{\sqsubset, \between\}, \emptyset$) and APS($\{<, \between\}, \emptyset$) are **NP**-complete and gave positive results by showing that APS($\{\between\}$, $\emptyset$) and APS($\{\between\}$,$\{\between\}$) are polynomial time solvable. Hence, the refinement we suggest shows that APS problem becomes hard when one considers sequences containing $\{\between, \alpha\}$-comparable arcs with $\alpha \neq \emptyset$. Therefore, crossing arcs alone do not imply APS hardness. It is of course a challenging problem to

Table 3. Complexity results after refinement of the complexity levels. ⋆: results from this paper.

APS								
R_1 \ R_2	$\{<,\sqsubset,\between\}$	$\{\sqsubset,\between\}$	$\{<,\between\}$	$\{\between\}$	$\{<,\sqsubset\}$	$\{\sqsubset\}$	$\{<\}$	$\emptyset$
$\{<,\sqsubset,\between\}$	**NP**-C [6]	**NP**-C ⋆	**NP**-C [12]	**NP**-C ⋆	**NP**-C [12]	**NP**-C ⋆	**NP**-C [12]	**NP**-C ⋆
$\{\sqsubset,\between\}$		**NP**-C ⋆	////	**NP**-C ⋆	////	**NP**-C ⋆	////	**NP**-C ⋆
$\{<,\between\}$			**NP**-C ⋆	**NP**-C ⋆	////	////	**NP**-C ⋆	**NP**-C ⋆
$\{\between\}$				$O(nm^2)$ ⋆	////	////	////	$O(nm^2)$ ⋆
$\{<,\sqsubset\}$					$O(nm)$ [11]	$O(nm)$ [11]	$O(nm)$ [11]	$O(nm)$ [11]
$\{\sqsubset\}$						$O(nm)$ [11]	////	$O(nm)$ [11]
$\{<\}$							$O(nm)$ [11]	$O(n+m)$ [11]
$\emptyset$								$O(n+m)$ [11]

further explore the complexity of the APS problem, and especially the parameterized views, by considering additional parameters such as the cutwidth or the depth of the arc structures.

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Towards optimally solving the longest common subsequence problem for sequences with nested arc annotations in linear time. In *Proc. of the 13th Symposium on Combinatorial Pattern Matching (CPM02)*, volume 2373 of *LNCS*, pages 99–114. Springer-Verlag, 2002.
2. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Computing the similarity of two sequences with nested arc annotations. *Theoretical Computer Science*, 312(2-3):337–358, 2004.
3. B. Billoud, M.-A. Guerrucci, M. Masselot, and J.S. Deutsch. Cirripede phylogeny using a novel approach: Molecular morphometrics. *Molecular Biology and Evolution*, 19:138–148, 2000.
4. G. Caetano-Anolls. Tracing the evolution of RNA structure in ribosomes. *Nucl. Acids. Res.*, 30:2575–2587, 2002.
5. W. Chaia and V. Stewart. RNA Sequence Requirements for NasR-mediated, Nitrate-responsive Transcription Antitermination of the Klebsiella oxytoca M5al nasF Operon Leader. *Journal of Molecular Biology*, 292:203–216, 1999.
6. P. Evans. *Algorithms and Complexity for Annotated Sequence Analysis*. PhD thesis, U. Victoria, 1999.
7. P. Evans. Finding common subsequences with arcs and pseudoknots. In *Proc. of the 10th Symposium Combinatorial Pattern Matching (CPM99)*, volume 1645 of *LNCS*, pages 270–280. Springer-Verlag, 1999.
8. A.D. Farris, G. Koelsch, G.J. Pruijn, W.J. van Venrooij, and J.B. Harley. Conserved features of Y RNAs revealed by automated phylogenetic secondary structure analysis. *Nucl. Acids. Res.*, 27:1070–1078, 1999.
9. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
10. D. Goldman, S. Istrail, and C.H. Papadimitriou. Algorithmic aspects of protein structure similarity. In *Proc. of the 40th Symposium of Foundations of Computer Science (FOCS99)*, pages 512–522, 1999.
11. J. Gramm, J. Guo, and R. Niedermeier. Pattern matching for arc-annotated sequences. In *Proc. of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS02)*, volume 2556 of *LNCS*, pages 182–193, 2002.
12. J. Guo. Exact algorithms for the longest common subsequence problem for arc-annotated sequences. Master's Thesis, Universitat Tubingen, Fed. Rep. of Germany, 2002.
13. K. Hellendoorn, P.J. Michiels, R. Buitenhuis, and C.W. Pleij. Protonatable hairpins are conserved in the 5'-untranslated region of tymovirus RNAs. *Nucl. Acids. Res.*, 24:4910–4917, 1996.
14. L. Hofacker, M. Fekete, C. Flamm, M.A. Huynen, S. Rauscher, P.E. Stolorz, and P.F. Stadler. Automatic detection of conserved RNA structure elements in complete RNA virus genomes. *Nucl. Acids. Res.*, 26:3825–3836, 1998.

15. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proc. 11th Symposium on Combinatorial Pattern Matching (CPM00)*, volume 1848 of *LNCS*, pages 154–165. Springer-Verlag, 2000.
16. V. Juan, C. Crain, and S. Wilson. Evidence for evolutionarily conserved secondary structure in the H19 tumor suppressor RNA. *Nucl. Acids. Res.*, 28:1221–1227, 2000.
17. G. Lancia, R. Carr, B. Walenz, and S. Istrail. 101 optimal PDB structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. In *Proceedings of the 5th ACM International Conference on Computational Molecular Biology (RECOMB01)*, pages 193–202, 2001.
18. S.W.M. Teunissen, M.J.M. Kruithof, A.D. Farris, J.B. Harley, W.J. van Venrooij, and G.J.M. Pruijn. Conserved features of Y RNAs: a comparison of experimentally derived secondary structures. *Nucl. Acids. Res.*, 28:610–619, 2000.
19. S. Vialette. Pattern matching over 2-intervals sets. In *Proc. 13th Annual Symposium Combinatorial Pattern Matching (CPM02)*, volume 2373 of *LNCS*, pages 53–63. Springer-Verlag, 2002.
20. S. Vialette. On the computational complexity of 2-interval pattern matching. *Theoretical Computer Science*, 312(2-3):223–249, 2004.
21. H.-Y. Wang and S.-C. Lee. Secondary structure of mitochondrial 12S rRNA among fish and its phylogenetic applications. *Molecular Biology and Evolution*, 19:138–148, 2002.
22. J. Wuyts, P. De Rijk, Y. Van de Peer, G. Pison, P. Rousseeuw, and R. De Wachter. Comparative analysis of more than 3000 sequences reveals the existence of two pseudoknots in area V4 of eukaryotic small subunit ribosomal RNA. *Nucl. Acids. Res.*, 28:4698–4708, 2000.
23. K. Zhang, L. Wang, and B. Ma. Computing the similarity between RNA structures. In *Proc. 10th Symposium on Combinatorial Pattern Matching (CPM99)*, volume 1645 of *LNCS*, pages 281–293. Springer-Verlag, 1999.
24. M. Zuker. RNA folding. *Meth. Enzymology*, 180:262–288, 1989.

Profiling and Searching for RNA Pseudoknot Structures in Genomes

Chunmei Liu[1], Yinglei Song[1], Russell L. Malmberg[2], and Liming Cai[1]

[1] Department of Computer Science, University of Georgia, Athens GA 30602, USA
{chunmei, song, cai}@cs.uga.edu
[2] Department of Plant Biology, University of Georgia, Athens GA 30602, USA
russell@plantbio.uga.edu

Abstract. We developed a new method that can profile and efficiently search for pseudoknot structures in noncoding RNA genes. It profiles interleaving stems in pseudoknot structures with independent Covariance Model (CM) components. The statistical alignment score for searching is obtained by combining the alignment scores from all CM components. Our experiments show that the model can achieve excellent accuracy on both random and biological data. The efficiency achieved by the method makes it possible to search for structures that contain pseudoknot in genomes of a variety of organisms.

1 Introduction

Searching genomes with computational models has become an effective approach for the identification of genes. During recent years, extensive research has been focused on developing computationally efficient and accurate models that can find novel noncoding RNAs and reveal their associated biological functions. Unlike the messenger RNAs that encode the amino acid residues of protein molecules, noncoding RNA molecules play direct roles in a variety of biological processes including gene regulation, RNA processing, and modification. For example, the human 7SK RNA binds and inhibits the transcription elongation factor P-TEFb [17][25] and the RNase P RNA processes the 5' end of precursor tRNAs and some rRNAs [7]. Noncoding RNAs include more than 100 different families [23]. Genome annotation based on models constructed from homologous sequence families could be a reliable and effective approach to enlarging the known families of noncoding RNAs.

The functions of noncoding RNAs are, to a large extent, determined by the secondary structures they fold into. Secondary structures are formed by bonded base pairs between nucleotides and may remain unchanged while the nucleotide sequence may have been significantly modified through mutations over the course of evolution. Profiling models based solely on sequence content such as Hidden Markov Model (HMM) [12] may miss structural homologies when directly used to search genomes for noncoding RNAs containing complex secondary structures. Models that can profile noncoding RNAs must include both the content and the structural information from the homologous sequences. The Covariance Model (CM) developed by Eddy and Durbin [6] extends the profiling HMM by allowing the coemission of paired nucleotides on certain

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 37–47, 2005.

states to model base pairs, and introduces bifurcation states to emit parallel stems. The CM is capable of modeling secondary structures comprised of nested and parallel stems. However, pseudoknot structures, where at least two structurally interleaving stems are involved, cannot be directly modeled with the CM and have remained computationally intractable for searching [1][13][14][18][19][20][21][24].

So far, only a few systems have been developed for profiling and searching for RNA pseudoknots. One example is ERPIN developed by Gautheret and Lambert [8][15]. ERPIN searches genomes by sequentially looking for single stem loop motifs contained in the noncoding RNA gene, and reports a hit when significant alignment scores are observed for all the motifs at their corresponding locations. Since ERPIN does not allow the presence of gaps when it performs alignments, it is computationally very efficient. However, alignments with no gaps may miss distant homologies and thus result in a lower sensitivity.

Brown and Wilson [2] proposed a more realistic model comprised of a number of Stochastic Context Free Grammar (SCFG) [3][22] components to profile pseudoknot structures. In their model, the interleaving stems in a pseudoknot structure are derived from different components; the pseudoknot structure is modeled as the intersection of components. The optimal alignment score of a sequence segment is computed by aligning it to all the components iteratively. The model can be used to search sequences for simple pseudoknot structures efficiently. However, a generic framework for modeling interleaving stems and carrying out the search was not proposed in their work. For pseudoknots with more complex structure, more than two SCFG components may be needed and the extension of the iterative alignment algorithm to k components may require $k!$ different alignments in total since all components are treated equally in their model.

In this paper, we propose a new method to search for RNA pseudoknot structures using a model of multiple CMs. Unlike the model of Brown and Wilson, we use independent CM components to profile the interleaving stems in a pseudoknot. Based on the model, we have developed a generic framework for modeling interleaving stems of pseudoknot structures; we propose an algorithm that can efficiently assign stems to components such that interleaving stems are profiled in different components. The components with more stems are associated with higher weights in determining the overall conformation of a sequence segment. In order to efficiently perform alignments of the sequence segment to the model, instead of iteratively aligning the sequence segment to the CM components, our searching algorithm aligns it to each component independently following the descending order of component weights. The statistical log-odds scores are computed based on the structural alignment scores of each CM component. *Stem contention* may occur such that two or more base pairs obtained from different components require the participation of the same nucleotide. Due to the conformational constraints inherently imposed by the CM components, stem contentions occur infrequently (less than 30%) and can be effectively resolved based on the conformational constraints from the alignment results on components with higher weight values. The algorithm is able to accomplish the search with a worst case time complexity of $O((k-1)W^3L)$ and a space complexity of $O(kW^2)$, where k is the number of CM components in the model, W and L are the size of the searching window and the length of the genome respectively.

We used the model to search for a variety of RNA pseudoknots inserted in randomly generated sequences. Experiments show that the model can achieve excellent sensitivity (SE) and specificity (SP) on almost all of them, while using only slightly more computation time than searching for pseudoknot-free RNA structures. We then applied the model and the searching algorithm to identify the pseudoknots on the 3' untranslated region in several RNA genomes from the corona virus family. An exact match between the locations found by our program and the real locations is observed. Finally, in order to test the ability of our program to cope with noncoding RNA genes with complex pseudoknot structures, we carried out an experiment where the complete DNA genomes of two bacteria were searched to find the locations of the tmRNA genes. The results show that our program identified the location with a reasonable amount of error (with a right shift of around 20 nucleotide bases) for one bacterial genome and for the other bacteria search was perfect. To the best of our knowledge, this is the first experiment where a whole genome of more than a million nucleotides is searched for a complex structure that contains pseudoknots.

2 Experiments and Results

To test the performance of the model, we developed a search program in C language and carried out searching experiments on a Sun/Solaris workstation. The workstation has 8 dual processors and 32GB main memory. We evaluated the accuracy of the program on both real genomes and randomly generated sequences with a number of RNA pseudoknot structures inserted. The RNAs we choose to test the model are shown in Table 1. Model training and testing are based on the multiple alignments downloaded from the Rfam database [10]. For each RNA pseudoknot, we divided the available data into a training set and a testing set, and the parameters used to model it are estimated based on multiple structural alignments among $5 - 90$ homologous training sequences with a pairwise identity less than 80%. The emission probabilities of all nucleotides for a given state in a CM component are estimated by computing their frequencies to appear in the corresponding column in the multiple alignment of training sequences; transition probabilities are computed similarly by considering the rel-

Table 1. Information on training sequences used for the estimation of model parameters

RNA	Number of training sequences	Number of nucleotides	Pseudocount
tmRNA−pk12	36	$130 - 250$	1.5
tmRNA−pk34	89	$90 - 120$	2.4
srpRNA	24	$30 - 50$	1.2
telomerase−vert	13	$90 - 200$	0.9
corona−pk3	14	$60 - 70$	0.9
HDV−ribozyme	15	$90 - 100$	1.0
tombus−3−IV	17	$90 - 100$	1.0
alpha−RBS	9	$100 - 120$	0.8
antizyme−FSE	13	$50 - 60$	0.9
IFN−gamma	5	$160 - 180$	0.6

Table 2. The performance of the model on different RNA pseudoknots inserted into a background (of 10^5 nucleotides) randomly generated with different C+G concentrations. TN is the total number of pseudoknotted sequence segments inserted; CI is the number of sequence segments correctly identified by the program (with a positional error less than ± 3 bases); NH is the number of sequence segments returned by the program; SE and SP are sensitivity and specificity respectively. The thresholds of log-odds score are predetermined using the Z-score value of 4.0.

RNA	TN	CI	NH	SE(%)	SP(%)	Running time(hr)	Background C+G (%)
tmRNA−pk12	25	20	24	80.0	83.3	56.33	57.0
tmRNA−pk34	27	26	31	96.0	84.0	59.36	57.0
srpRNA	29	13	16	44.8	81.3	4.79	57.0
telomerase−vert	14	14	15	100.0	93.3	68.83	57.0
corona−pk3	37	37	39	100.0	94.8	2.89	57.0
HDV−ribozyme	37	37	37	100.0	100.0	6.54	57.0
tombus−3−IV	13	13	13	100.0	100.0	15.45	57.0
alpha−RBS	24	24	25	100.0	96.0	27.85	57.0
antizyme−FSE	28	28	28	100.0	100.0	0.94	57.0
IFN−gamma	10	10	10	100.0	100.0	31.24	57.0
tmRNA−pk12	24	24	25	100.0	96.0	55.57	67.0
tmRNA−pk34	27	27	30	100.0	90.0	56.42	67.0
srpRNA	25	17	19	68.0	89.4	4.76	67.0
telomerase−vert	13	13	14	100.0	92.9	67.80	67.0
corona−pk3	33	33	34	100.0	97.1	2.90	67.0
HDV−ribozyme	37	37	37	100.0	100.0	6.52	67.0
tombus−3−IV	20	20	20	100.0	100.0	16.63	67.0
alpha−RBS	18	18	18	100.0	100.0	27.79	67.0
antizyme−FSE	28	28	29	100.0	96.6	0.94	67.0
IFN−gamma	10	10	10	100.0	100.0	33.15	67.0
tmRNA−pk12	26	26	29	100.0	90.0	55.45	77.0
tmRNA−pk34	25	25	33	100.0	75.7	53.55	77.0
srpRNA	29	22	23	75.9	95.7	4.78	77.0
telomerase−vert	16	16	16	100.0	100.0	66.07	77.0
corona−pk3	37	37	37	100.0	100.0	3.13	77.0
HDV−ribozyme	37	37	37	100.0	100.0	6.57	77.0
tombus−3−IV	20	20	20	100.0	100.0	16.94	77.0
alpha−RBS	22	22	22	100.0	100.0	28.86	77.0
antizyme−FSE	28	28	28	100.0	100.0	0.96	77.0
IFN−gamma	10	10	10	100.0	100.0	32.55	77.0
tmRNA−pk12	24	24	25	100.0	96.2	55.09	87.0
tmRNA−pk34	27	27	28	100.0	96.4	52.39	87.0
srpRNA	26	25	25	96.2	100.0	4.81	87.0
telomerase−vert	17	17	17	100.0	100.0	70.60	87.0
corona−pk3	37	37	37	100.0	100.0	3.17	87.0
HDV−ribozyme	37	37	37	100.0	100.0	6.64	87.0
tombus−3−IV	20	20	20	100.0	100.0	16.94	87.0
alpha−RBS	24	23	23	95.8	100.0	29.08	87.0
antizyme−FSE	26	26	26	100.0	100.0	0.94	87.0
IFN−gamma	10	10	10	100.0	100.0	32.84	87.0

ative frequencies for different types of transitions that occur between the corresponding consecutive columns in the alignment. Pseudocounts, dependent on the number of training sequences, are included to prevent overfitting of the model to the training data.

To measure the sensitivity and specificity of the searching program within a reasonable amount of time, for each selected pseudoknot structure, we selected $10 - 40$ sequence segments from the set of testing data and inserted them into each of the randomly generated sequences of 10^5 nucleotides. In order to test whether the model is sensitive to the base composition of the background sequence, we varied the C+G concentration in the random background. The program computes the log-odds, the logarithmic ratio of the probability of generating sequence segment s by the null (random) model R to that by our model M. It reports a hit when the Z-score of s is greater than 4.0. The computation of Z-scores requires knowing the mean and standard deviation for the distribution of log-odd scores of random sequence segments; both of them can be determined with methods similar to the ones introduced by Klein and Eddy [11] before the search starts.

As can be seen in Table 2, the program correctly identifies more than 80% of inserted sequence segments with excellent specificity in most of the experiments. The only exception is the srpRNA, where the program misses more than 50% inserted sequence segments in one of the experiments. The relatively lower sensitivity in that particular experiment can be partly ascribed to the fact that the pseudoknot structure of srpRNA contains fewer nucleotides; thus its structural and sequence patterns have a larger probability to occur randomly. The running time for srpRNA, however, is also significantly shorter than that needed by most of other RNA pseudoknots due to the smaller size of the model. Additionally, while the alpha−RBS pseudoknot has a more complex structure and three CM components are needed to model it, our searching algorithm efficiently identifies more than 95% of the inserted pseudoknots with high specificities. A higher C+G concentration in the background does not adversely affect the specificity of the model; it is evident from Table 2 that the program achieves better overall performance in both sensitivity and specificity in a background of higher C+G concentrations. We therefore conjecture that the specificity of the model is partly determined by the base composition of the genome and is improved if the base composition of the target gene is considerably different from its background.

To test the accuracy of the program on real genomes, we performed experiments to search for particular pseudoknot structures in the genomes for a variety of organisms. Table 3 shows the genomes on which we have searched with our program and the locations annotated for the corresponding pseudoknot structures. The program successfully identified the exact locations of known 3'UTR pseudoknot in four genomes from the family of corona virus. This pseudoknot was recently shown to be essential for the replication of the viruses in the family [9].

In addition, the genomes of the bacteria, *Haemophilus influenzae* and *Neisseria meningitidis MC58*, were searched for their tmRNA genes. The *Haemophilus influenzae* DNA genome contains about 1.8×10^6 nucleotides and *Neisseria meningitidis MC58* DNA genome contains about 2.2×10^6 nucleotides. The tmRNA functions in the trans-translation process to add a C-terminal peptide tag to the incomplete protein product of

Table 3. The results obtained with our searching program on the genomes of a variety of organisms. GA is the accession number of the genome; RL specifies the real location of the pseudoknot structure in the genome; SL is the one returned by the program; RT is the running time needed to perform the searching in hours; GL is the length of the genome in its number of bases. The genome of Haemophilus searched in our experiment is the reversed complementary DNA strand.

GA	Organism	ncRNA	RL	SL	RT(hr)	GL(bs)
NC000907	Haemophilus	tmRNA	$472210 - 472575$	$472177 - 472542$	170.00	1.83×10^6
NC003112	Neisseria meningitidis	tmRNA	1241197– 1241559	1241197– 1241559	170.00	2.2×10^6
NC003045	Bovine CoronaVirus	3'UTR pk	$30798 - 30859$	$30798 - 30859$	1.24	31028
NC002645	Human CoronaVirus	3'UTR pk	$27063 - 27125$	$27063 - 27125$	1.12	27317
NC001846	Murine HepatitusVirus	3'UTR pk	$31092 - 31153$	$31092 - 31153$	1.27	31357
NC003436	Porcine DiarrheaVirus	3'UTR pk	$27820 - 27882$	$27820 - 27882$	1.17	28033

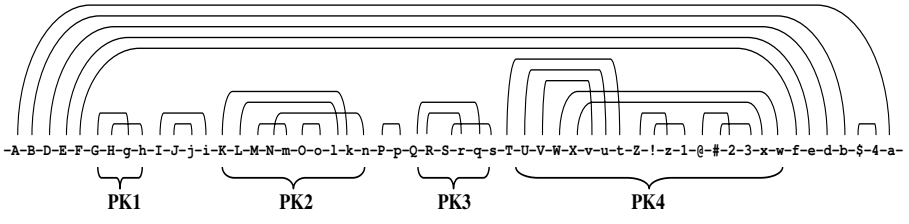


Fig. 1. Diagram of the pairing regions on the tmRNA gene. Upper case letters indicate base sequences that pair with the corresponding lower case letters. The four pseudoknots constitute the central part of the tmRNA gene and are called Pk1, Pk2, Pk3, Pk4 respectively.

a defective mRNA [16]. The central part of the secondary structure of tmRNA molecule consists of four pseudoknot structures. Figure 1 shows the pseudoknot structures on the tmRNA molecule.

In order to search the bacterial DNA genomes efficiently, the combined pseudoknots 1 and 2 were used to search the genome first; the program searches for the whole tmRNA gene only in the region around the locations where a hit for Pk1 and Pk2 is detected. We cut the genome into segments with shorter lengths (around 10^5 nucleotide bases for each), and ran the program in parallel on ten of them in two rounds. The result for *Neisseria meningitidis MC58* shows that we successfully identified the exact locations of tmRNA. However, the locations of tmRNA obtained for *Haemophilus influenzae* have a shift of around 20 nucleotides with respect to its real location (7% of the length of the tmRNA). This slight error can probably be ascribed to our "hit-and-extend" searching strategy to resolve the difficulty arising from the complex structure and the relatively much larger size of tmRNA genes; positional errors may occur during different searching stages and accumulate to a significant value. Our experiment on the

DNA genomes also demonstrates that, for each genome, it is very likely there is only one tmRNA gene in it, since our program found only one significant hit. To our knowledge, this is the first computational experiment where a whole genome of more than a million nucleotides was successfully searched for a complex structure that contains pseudoknot structures.

3 Models and Algorithms

The Covariance Model (CM) proposed by Eddy and Durbin [6][5] can effectively model the base pairs formed between nucleotides in an RNA molecule. Similarly to the emission probabilities in HMMs, the emission probabilities in the CM for both unpaired nucleotides and base pairs are positional dependent. The profiling of a stem hence consists of a chain of consecutive emissions of base pairs. Parallel stems on the RNA sequence are modeled with bifurcation transitions where a bifurcation state is split into two states. The parallel stems are then generated from the transitions starting with the two states that result respectively.

The genome is scanned by a window with an appropriate length. Each location of the window is scored by aligning all subsequence segments contained in the window to the model with the CYK algorithm. The maximum log-odds score of them is determined as the log-odds score associated with the location. A hit is reported for a location if the computed log-odds score is higher than a predetermined threshold value.

Pseudoknot structures are beyond the profiling capability of a single CM due to the inherent context sensitivity of pseudoknots. Models for pseudoknot structures require a mechanism for the description of their interleaving stems. Previous work by Brown and Wilson [2] and Cai *et al.* [4] has modeled the pseudoknot structures with grammar components that intersect or cooperatively communicate. A similar idea is adopted in this work; a number of independent CM components are combined to resolve the difficulty in profiling that arises from the interleaving stems. Interleaving stems are profiled in different CM components and the alignment score of a sequence segment is determined based on a combination of the alignment scores on all components.

However, the optimal conformations from the alignments on different components may violate some of the conformational constraints that a single RNA sequence must follow. For example, a nucleotide rarely forms two different base pairs simultaneously with other nucleotides in an RNA molecule. This type of restriction is not considered by the independent alignments carried out in our model and thus may lead to erroneous searching results if not treated properly. In our model, *stem contention* may occur. We break the contention by introducing different priorities to components; base pairs determined from components with the highest priority win the contention. We hypothesize that, biochemically, components profiling more stems are likely to play more dominant roles in the formation of the conformation and are hence assigned higher priority weights.

3.1 Model Generation

In order to profile the interleaving stems in a pseudoknot structure with independent CM components, we need an algorithm that can partition the set of stems on the RNA sequence into a number of sets comprised of stems that mutually do not interleave. Based

on the consensus structure of the RNA sequence, an undirected graph $G = (V, E)$ can be constructed where V, the set of vertices in G, consists of all stems on the sequence. Two vertices are connected with an edge in G if the corresponding stems are in parallel or nested. The set of vertices V needs to be partitioned into subsets such that the subgraph induced by each subset forms a clique.

We use a greedy algorithm to perform the partition. Starting with a vertex set S initialized to contain a arbitrarily selected vertex, the algorithm iteratively searches the neighbors of the vertices in S and computes the set of vertices that are connected to all vertices in S. It then randomly selects one vertex v that is not in S from the set and modifies S by assigning v to S. The algorithm outputs S as one of the subsets in the partition when S can not be enlarged and randomly selects an unassigned vertex and repeats the same procedure. It stops when every vertex in G has been included in a subset. Although the algorithm does not minimize the number of subsets in the partition, our experiments show that it can efficiently provide optimal partitions of the stems on pseudoknot structures of moderate structural complexity.

The CM components in the profiling model are generated and trained based on the partition of the stems. The stems in the same subset are profiled in the same CM component. For each component, the parameters are estimated by considering the consensus structure formed by the stems in the subset only.

3.2 Searching Algorithm

The optimal alignments of a sequence segment to the CM components are computed with the dynamic programming based CYK algorithm. As we have mentioned before, higher priority weights are assigned to components with more stems profiled. The component with the maximum number of stems thus has the maximum weight and is the *dominant component* in the model. The algorithm performs alignments in the descending order of component weights. It selects the sequence segment that maximizes the log-odds score from the dominant component. The alignment scores and optimal conformations of this segment on other components are then computed and combined to obtain the overall log-odds score for the segment's position on the genome.

More specifically, we assume that the model contains k CM components $M_0, M_1, ..., M_{k-1}$ in descending order of component weights. The algorithm considers all possible sequence segments s_d that are enclosed in the window and uses Equation (1) to determine the sequence segment s to be the candidate for further consideration, where W is the length of the window used in searching, and Equation (2) to compute the overall log-odds score for s. We use sm_i to denote the parts of s that are aligned to the stems profiled in CM component M_i. Basically, $Log\_odds(sm_i|M_i)$ accounts for the contributions from the alignment of sm_i to M_i. The log-odds score of sm_i is counted in both M_0 and M_i and must be subtracted from the sum.

$$s = \arg \max_{0<|s_d|<W} \{Log\_odds(s_d|M_0)\}. \quad (1)$$

$$Log\_odds(s|M) = Log\_odds(s|M_0) + \sum_{i=1}^{k-1} \sum_{sm_i \in M_i} (Log\_odds(sm_i|M_i) - Log\_odds(sm_i|M_0)). \quad (2)$$

3.3 Stem Contention

The conformations corresponding to the optimal alignments of a sequence segment to all CM components are obtained by tracing back the dynamic programming matrices and checking to ensure that no stem contention occurs. Since each nucleotide in the sequence is represented with a state in a CM component, the CM inherently imposes constraints on the optimal conformations of sequence segments aligned to it. We hence expect that stem contention occurs with a low frequency. In order to verify this intuition, we tested the model on sequences randomly generated with different base compositions and evaluated the frequencies of stem contentions for pseudoknot structures on which we have performed an accuracy test; the results are shown in Figure 2.

The presence of stem contention increases the running time of the algorithm, because the alignment of one of the involved components must be recomputed to resolve the contention. Based on the assumption that components with more stems contribute more to the stability of the optimal conformation, we resolve the contention in favor of such components. We perform recomputation on the component with a lower number of stems by incorporating conformational constraints inherited from components with more stems into the alignment algorithm, preventing them from forming the contentious stems.

Specifically, we assume that stem $S_j \in M_i$ and stem contention occurs between S_j and other stems profiled in M_{i-1}; the conformational constraints from the component M_{i-1} are in the format of (l_1, l_2) and (r_1, r_2). In other words, to avoid the stem con-

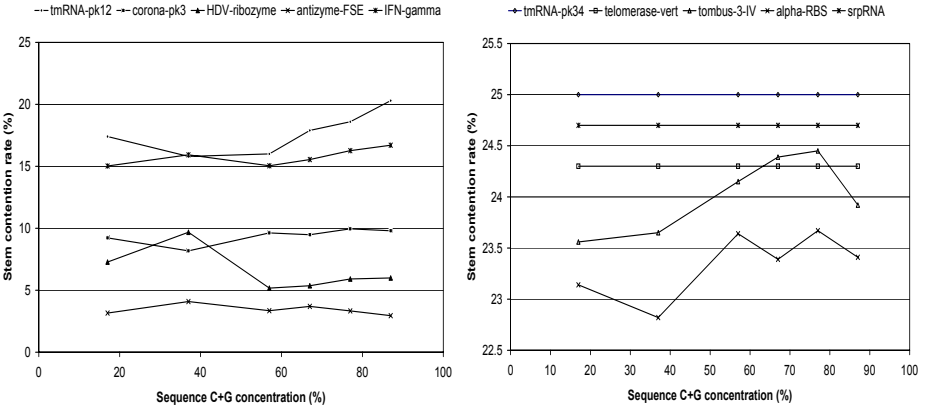


Fig. 2. 4000 random sequences were generated at each given base composition and aligned to the corresponding profiling model. The sequences are of about the same length as the length of the pseudoknot structure. The stem contention rates for each pseudoknot structure were measured and plotted. They were the ratio of the number of random sequences in which stem contentions occurred to the number of total random sequences. Left: plots of profiling models observed to have a stem contention rate lower than 20%, right: plots of these with slightly higher stem contention frequencies. The experimental results demonstrate that, in all pseudoknots where we have performed accuracy tests, stem contention occurs with a rate lower than 30% and is insensitive to the base composition of sequences.

tention, the left and right parts of the stem must be the subsequences of indices (l_1, l_2) and (r_1, r_2) respectively. The dynamic programming matrices for S_j are limited to the rectangular region that satisfies $l_1 \leq s \leq l_2$ and $r_1 \leq t \leq r_2$.

The stem contention frequency depends on the conformational flexibilities of the components in the covariance model. More flexibilities in conformation may improve the sensitivity of the model but cause higher contention frequency and thus increase the running time for the algorithm. In the worst case, recomputation is needed for all non-dominant components in the model and the time complexity of the algorithm becomes $O((k-1)W^3L)$, where k is the number of components in the model, W and L are the window length and the genome length respectively.

4 Conclusions and Future Work

In this paper, we have introduced a new model that serves as the basis for a generic framework that can efficiently search genomes for the noncoding RNAs with pseudoknot structures. Within the framework, interleaving stems in pseudoknot structures are modeled with independent CM components and alignment is performed by aligning sequence segments to all components following the descending order of their weight values. Stem contention occurs with a low frequency and can be resolved with a dynamic programming based recomputation. The statistical log-odds scores are computed based on the alignment results from all components. Our experiments on both random and biological data demonstrate that the searching framework achieves excellent performance in both accuracy and efficiency and can be used to annotate genomes for noncoding RNA genes with complex secondary structures in practice.

We were able to search a bacterial genome for a complete structure with a pseudoknot in about one week on our Sun workstation. It would be desirable to improve our algorithm so that we could search larger genomes and databases. The running time, however, could be significantly shortened if a filter can be designed to preprocess DNA genomes and only the parts that pass the filtering process are aligned to the model. Alternatively, it may be possible to devise alternative profiling methods to the covariance model that would allow faster searches.

References

1. T. Akutsu, "Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots.", *Discrete Applied Mathematics*, 104: 45-62, 2000.
2. M. Brown and C. Wilson, "RNA Pseudoknot Modeling Using Intersections of Stochastic Context Free Grammars with Applications to Database Search.", *Pacific Symposium on Biocomputing*, 109-125, 1995.
3. M. Brown, "Small subunit ribosomal RNA modeling using stochastic context-free grammars.", *Proc. of Int. Conf. Intel. Syst. Mol. Biol.*, 56: 57-66, 2000.
4. L. Cai, R. L. Malmberg, and Y. Wu, "Stochastic Modeling of Pseudoknot Structures: A Grammatical Approach.", *Bioinformatics*, 19, $i66 - i73$, 2003.
5. R. Durbin, S. R. Eddy, A. Krogh, and G. J. Mitchison, "Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.", *Cambridge University Press*, 1998.

6. S. Eddy and R. Durbin, "RNA sequence analysis using covariance models.", *Nucleic Acids Research*, 22: 2079-2088, 1994.
7. D. N. Frank and N. R. Pace, "Ribonuclease P: unity and diversity in a tRNA processing ribozyme.", *Annu Rev Biochem.*, 67: 153-180, 1998.
8. D. Gautheret and A. Lambert, "Direct RNA motif definition and identification from multiple sequence alignments using secondary structure profiles.", *Journal of Molecular Biology*, 313: 1003-1011, 2001.
9. S. J. Geobel, B. Hsue, T. F. Dombrowski, and P. S. Masters, "Characterization of the RNA components of a Putative Molecular Switch in the 3' Untranslated Region of the Murine Coronavirus Genome.", *Journal of Virology*, 78: 669-682, 2004.
10. S. Griffiths-Jones, A. Bateman, M. Marshall, A. Khanna, and S. R. Eddy, "Rfam: an RNA family database.", *Nucleic Acids Research*, 31: 439-441, 2003.
11. R. J. Klein, S. R. Eddy, "RSEARCH: Finding Homologs of Single Structured RNA Sequences.", *BMC Bioinformatics*, 4:44, 2003.
12. A. Krogh, M. Brown, IS. Mian, K. Sjolander, and D. Haussler, "Hidden Markov models in computational biology. Applications to protein modeling.", *Journal of Molecular Biology*, 235: 1501-1531, 1994.
13. D. Lee, K. Han, "Prediction of RNA Pseudoknots-Comparative Study of Genetic Algorithms.", *Genome Informatics*, 13: 414-415, 2002.
14. R. B. Lyngso and C. N. S. Pederson, "RNA pseudoknot prediction in energy based models.", *Journal of Computational Biology*, 7: 409-428, 2000.
15. T. Macke, D. Ecker, R. Gutell, and D. Gautheret, D. Case, R. Sampath, "RNAMotif, an RNA secondary structure definition and search algorithm.", *Nucleic Acids Research*, 29: 4724-4735, 2001.
16. N. Nameki, B. Felden, J. F. Atkins, R. F. Gesteland, H. Himeno, A. Muto, "Functional and structural analysis of a pseudoknot upstream of the tag-encoded sequence in E. coli tmRNA.", *Journal of Molecular Biology*, 286(3): 733-744, 1999.
17. V. T. Nguyen, T. Kiss, A. A. Michels, O. Bensaude, "7SK small nuclear RNA binds to and inhibits the activity of CDK9/cyclin T complexes.", *Nature* 414: 322-325, 2001.
18. J. Reeder and R. Giegeritch, "Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics.", *BMC Bioinformatics*, 5: 104, 2004.
19. E. Rivas and S. Eddy, "The language of RNA: a formal grammar that includes pseudoknots.", *Bioinformatics*, 16: 334-340, 2000.
20. E. Rivas and S. Eddy, "A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots.", *Journal of Molecular Biology*, 285: 2053-2068, 1999.
21. J. Ruan, G. D. Stormo, and W. Zhang, "An iterated loop matching approach to the prediction of RNA secondary structures with pseudoknots.", *Bioinformatics*, 20: 58-66, 2004.
22. Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjolander, R. C. Underwood, and D. Maussler, "Stochastic Context-Free Grammars for tRNA Modeling.", *Nucleic Acids Research*, 22: 5112-5120, 1994.
23. G. Storz, "An expanding universes of noncoding RNAs.", *Science*, 296(5571): 1260-1263, 2002.
24. Y. Uemura, A. Hasegawa, Y. Kobayashi, T. Yokomori, "Tree adjoining grammars for RNA structure prediction.", *Theoretical Computer Science*, 210: 277-303, 1999.
25. Z. Yang, Q. Zhu, K. Luo, and Q. Zhou, "The 7SK small nuclear RNA inhibits the Cdk9/cyclin T1 kinase to control transcription.", *Nature* 414: 317-322, 2001.

A Class of New Kernels Based on High-Scored Pairs of *k*-Peptides for SVMs and Its Application for Prediction of Protein Subcellular Localization

Zhengdeng Lei and Yang Dai*

Department of Bioengineering (MC063),
University of Illinois at Chicago,
851, South Morgan Street, Chicago, IL 60607, USA
{zlei2, yangdai}@uic.edu

Abstract. A class of new kernels has been developed for vectors derived from a coding scheme of the k-peptide composition for protein sequences. Each kernel defines the biological similarity for two mapped k-peptide coding vectors. The mapping transforms a k-peptide coding vector into a new vector based on a matrix formed by high BLOSUM scores associated with pairs of k-peptides. In conjunction with the use of support vector machines, the effectiveness of the new kernels is evaluated against the conventional coding scheme of k-peptide ($k \leq 3$) for the prediction of subcellular localizations of proteins in Gram-negative bacteria. It is demonstrated that the new method outperforms all the other methods in a 5-fold cross-validation.

Keywords: Protein subcellular localization, BLOSUM matrix, kernel, support vector machine, Gram-negative bacteria.

1 Introduction

Advances in genome sequencing and proteomics are generating enormous numbers of genes and proteins. Accordingly, the development of automated systems for the annotation of protein structure and function has become extremely important. Since many cellular functions are compartmentalized in specific regions of a cell, subcellular localization of a protein is biologically highlighted as a key element in understanding its function. Specific knowledge of the subcellular location can direct further experimental study of proteins.

Methods and systems have been developed during the last decade for the predictive task of protein localization. Machine learning methods such as Artificial Neural Networks, the k-nearest neighbor method, and Support Vector Machines (SVMs) have been utilized in conjunction with various methods of feature extraction for protein sequences. Most of the early approaches employed

* Corresponding author.

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 48–58, 2005.

the amino acid and di-peptide compositions [7,12,27,28] to represent sequences. These methods may miss information on sequence order and inter-relationships among amino acids. In order to overcome these shortcomings, it has been shown that motifs, frequent-subsequences, functional domains, and other useful features, which are obtained from various databases (SMART, InterPro, PROSITE) or extracted using Hidden Markov Models, Fourier Transform, and other data mining techniques, can be used for the representation of protein sequences for the prediction of subcellular localizations [2,3,6,15,29,30]. Methods have also been developed based on the use of the N-terminal sorting signals [1,5,10,21,24,25,26] and sequence homology searching [23].

Most robust methods adopt an integrative approach by combining several methods, each of which may be a suitable predictor for a specific localization or a generic predictor for all localizations. PSORT is an example of such successful system. Developed by Nakai and Kanehisa [25], PSORT, recently upgraded to PSORT II [11,24], is an expert system that can distinguish between different subcellular localizations in eukaryotic cells. It also has a dedicated subsystem PSORT-B for bacterial sequences [8].

Several recent studies [19,31], however, have indicated that a predicting system based on the use of a generalized k-peptide composition or sequence homology could obtain similar or better performance compared to that of the integrated system PSORT-B. The outcome from our work supports these findings.

In this study, a new similarity measurement for protein sequences has been developed based on the use of high-scored pairs of k-peptides. It is the extension of the concept used in our previous work [16] for a fixed k value ($k = 3$). More specifically, each pair of k-peptides is assigned a score based on a BLOSUM matrix. A small portion of pairs with high scores is selected to retain their original scores in order to reduce noise and computational time. The remaining pairs are given zero scores. The reassigned score associated with each pair of k-peptides is then considered as an entry in a matrix D_k, which is named as the matrix of high-scored pairs of k-peptides. When $k = 1$, this matrix is the same as the BLOSUM matrix, except that the entries with negative values are replaced by zeroes. When $k \geq 2$, each entry is the BLOSUM score corresponding to a pair of k-peptides with negative value being replaced by zero. Each protein sequence is first coded by its k-peptide composition. Then each k-peptide coding vector $\boldsymbol{x}_k$ is mapped to another vector $D_k\boldsymbol{x}_k$, and the similarity between the sequences is measured by those mapped vectors. That is, the kernel is defined based on these mapped vectors.

The new kernels combined with SVMs are evaluated against the conventional coding scheme of k-peptide ($k \leq 3$) composition for the prediction of subcellular localizations for proteins obtained from Gram-negative bacteria [8]. It is demonstrated by the result of a 5-fold cross-validation that the new kernel method outperforms significantly the coding methods based on the conventional k-peptide composition.

2 Method

This section introduces a new kernel for the coding vectors derived from the k-peptide compositions of protein sequences. This coding scheme based on the k-peptide composition for $k \leq 2$ has been used for the prediction of subcellular localizations [12,27,31], but has never been directly evaluated for $k = 3$. Below a short description of SVMs is presented.

2.1 Support Vector Machines

Suppose that a set of m training points $\boldsymbol{x}_i$ $(1 \leq i \leq m)$ in an n-dimensional space is given. Each point $\boldsymbol{x}_i$ is labeled by $y_i \in \{1, -1\}$ denoting the membership of the point. An SVM is a learning method for binary classification. Using a nonlinear transformation ϕ, it maps the data to a high dimensional feature space in which a linear classification is performed. It is equivalent to solving the quadratic optimization problem:

$$\begin{array}{ll} \min\limits_{\boldsymbol{w},b,\xi_1,\ldots,\xi_m} & \frac{1}{2}\boldsymbol{w}\cdot\boldsymbol{w} + C\sum\limits_{i=1}^{m}\xi_i \\ \text{subject to} & y_i(\phi(\boldsymbol{x}_i)\cdot\boldsymbol{w}+b) \geq 1-\xi_i \quad (i=1,...,m), \\ & \xi_i \geq 0 \quad (i=1,...,m), \end{array} \tag{1}$$

where C is a parameter. The decision function is defined as $f(\boldsymbol{x}) = \text{sign}(\phi(\boldsymbol{x}) \cdot \boldsymbol{w} + b)$, where $\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i \phi(\boldsymbol{x}_i)$ and α_i $(i = 1, ..., m)$ are constants determined by the dual problem of the optimization defined above.

For any pair of mappings $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$, the kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is defined as a dot product of $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$, i.e.,

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j). \tag{2}$$

The kernel function is essentially a measurement of similarity for the mapped points in terms of their inner products. The matrix $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is called the kernel matrix. The decision function can be represented by using the kernel function:

$$f(\boldsymbol{x}) = \text{sign}(\sum_{i=1}^{m} \alpha_i \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}) + b) = \text{sign}(\sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b). \tag{3}$$

Typical kernel functions are, for example, polynomial kernel $(\boldsymbol{x}_i \cdot \boldsymbol{x}_j + a)^d$ $(d \geq 1)$ and the radial basis kernel $\exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$. In most of these cases, the corresponding nonlinear mappings ϕ are not known explicitly, although their existence is guaranteed. For other details of SVMs refer to [4].

2.2 Sequence Coding Schemes and a Class of New Kernels Based on High-Scored Pairs of *k*-Peptides

The effectiveness of the coding schemes for protein sequences based on the k-peptide compositions or their variations has been demonstrated in the prediction

of subcellular localizations, combining with machine learning tools such as neural networks and support vector machines [12,23,27,31]. If $k = 1$, the k-peptide composition reduces to the amino acid composition, and if $k = 2$, the k-peptide composition gives the di-peptide composition. When k becomes larger, the k-peptide composition will encompass more global sequence information, but at the same time, such a coding scheme becomes less attractive from the computational viewpoint.

In order to code a sequence, a window with a length of k is moved along the sequence from the first amino acid to the kth amino acid from the end. Every k-letter pattern that appears in the window is recorded with an increment of 1 in the corresponding entry of the vector. The final vector is normalized by dividing the number of window positions associated with that sequence. Upon the termination of this procedure, the vector provides the k-peptide composition of the sequence. Since the symbol "X" may appear in some sequences, it is added to the set of the original 20 symbols of the amino acids to give a total of 21. Therefore, vectors of 21, $21^2 = 441$ and $21^3 = 9261$ dimensions are required, respectively, for $k = 1, 2$, and 3 in this coding scheme.

However, a more sensitive and biologically relevant coding method would allow some degree of mismatch of amino acids in the k-peptide representation for $k \neq 1$. That is, the similarity should be large if two sequences share many similar k-peptides. This idea has been explored by Leslie *et al.* [17] for protein homology detection, and a set of mismatch kernels was developed. In their paper, the coding vector represents the occurrence of the corresponding k-peptides and its mismatched peptides in a protein sequence. In our work, the concept of mismatch kernel is explored in an implicit and different way. The similarity of two k-peptides is measured by the sum of BLOSUM scores between two residues at the same position.

In order to define the new kernel, we introduce a matrix in which each entry corresponds to the pairwise score of two k-peptides. For example, the scores are 12 for an AAA-AAA pair, 11 for an AAY-ACY pair, and 6 for a TVW-TVR pair, if the BLOSUM62 matrix is used. Since the majority of all possible pairs is associated with lower scores, the elimination of those pairs can reduce noise that may confuse the prediction. In addition, this procedure also reduces training time. Accordingly, only a very small portion of the entries corresponding to high-scored pairs is kept given a proper threshold, and the other entries are replaced by 0 in the matrix. The resulting matrix is called *the matrix of high-scored pairs of k-peptides*, and is denoted as D_k. The new kernel $k(\cdot, \cdot)$ is then defined as

$$k(\boldsymbol{x}_k^i, \boldsymbol{x}_k^j) = \exp(-\gamma \| D_k \boldsymbol{x}_k^i - D_k \boldsymbol{x}_k^j \|^2) \tag{4}$$

for the radial basis functions; or

$$k(\boldsymbol{x}_k^i, \boldsymbol{x}_k^j) = (D_k \boldsymbol{x}_k^i \cdot D_k \boldsymbol{x}_k^j + a)^d, \quad d \geq 1 \tag{5}$$

for polynomial functions. Basically, the similarity is measured between the transformed vectors $D_k \boldsymbol{x}_k^i$ and $D_k \boldsymbol{x}_k^j$, instead of that between the original k-peptide coding vectors $\boldsymbol{x}_k^i$ and $\boldsymbol{x}_k^i$.

The example in Fig. 1 describes the coding vectors obtained from the two methods for two short amino acids sequences AAACY and AACCY: $\boldsymbol{x}_3^1$ and $\boldsymbol{x}_3^2$ are based on the tri-peptide composition; and $D_3\boldsymbol{x}_3^1$ and $D_3\boldsymbol{x}_3^2$. For the tri-peptide composition, the vectors $\boldsymbol{x}_3^1$ and $\boldsymbol{x}_3^2$ share one common tri-peptide "AAC", which is the entry 2 in the vectors. However, the transformed vectors $D_3\boldsymbol{x}_3^1$ and $D_3\boldsymbol{x}_3^2$ have many non-zero common entries, such as 2, 16, 23, 24, 26, 28, etc (see boldfaced numbers in Fig.1). This implies that the transformation can capture similarity even if the two sequences do not share many exactly matched tri-peptides.

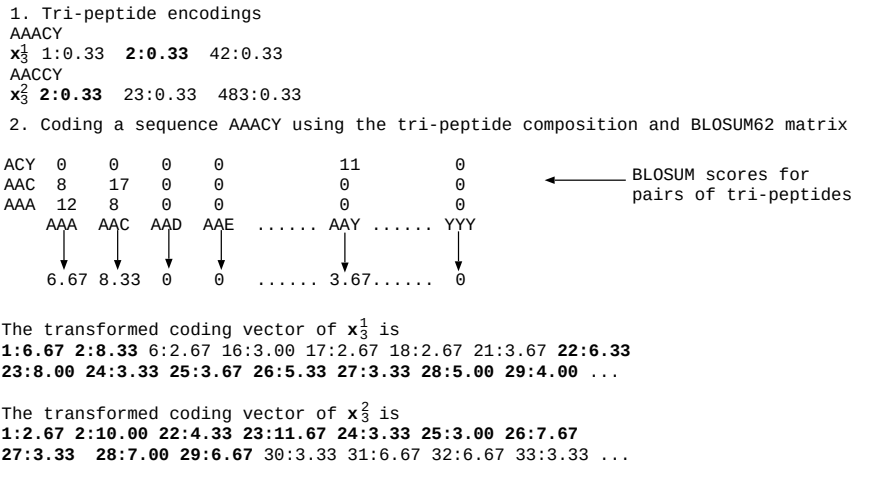


Fig. 1. The coding vectors for sequences AAACY and AACCY based on the tri-peptide composition and the transformed vectors based on high-scored pairs of tri-peptides. The representation of coding vectors follows the sparse format of SVMLight [14], i.e., the numbers appearing in the format of **vector index : score**. The shared elements between two sequences are boldfaced.

It is noted that the size of the matrix D_k for $k = 3$ is 9261×9261. However, after score thresholding, very few non-zero entries in the matrix are kept. Therefore, the matrix is represented using a sparse data structure to ensure the efficiency of computation. The selection of the high-scored pairs of k-peptides is virtually filtering the k-peptides sharing more residues in common. In addition, the procedure also retains those pairs with high similarity BLOSUM scores between the residues.

3 Experimental Results and Discussion

In order to evaluate the performance of our new kernels on the prediction of protein subcellular localization for different values of $k = 1, 2, 3$, a set of proteins from Gram-negative bacteria was used. In addition, the computation with

the conventional k-peptide ($k = 1, 2, 3$) coding scheme was also performed for comparison.

3.1 Dataset

The set of proteins from Gram-negative bacteria used in the evaluation of PSORT-B [8] was considered (available at `http://www.psort.org/`) in this experiment. It consists of 1443 proteins with experimentally determined localizations. The dataset comprises 1302 proteins resident at a single localization site: 248 cytoplasmic, 268 inner membrane, 244 periplasmic, 352 outer membrane, and 190 extra cellular; it additionally contains a set of 141 proteins resident at multiple localization sites: 14 cytoplasmic/inner membrane, 50 inner membrane/periplasmic, and 77 outer membrane/extracellular. In our experiment, we considered only the 1302 proteins possessing a single localization.

3.2 Experiments and Results

The BLOSUM62 matrix was used for the assignment of scores to pairs of k-peptides. The threshold for high-scored pairs was 0 for $k = 1, 2$; and 8 for $k = 3$. The nonzero entries account for about 1.3% of the entries in matrix D_3. In order to ease the computational burden, the 2000 top scored entries from a transformed vector $D_3\boldsymbol{x}_3$ were further selected to form the input vector for SVMs. The threshold 8 and the number 2000 were determined empirically from the preliminary study to ensure good performance and fast training.

The experiment was carried out with a 5-fold cross-validation (CV) for each specific localization. Each time, the relevant dataset consisting of the proteins with the specific localizations was designated as the positive set; and the remainder of the proteins was denoted as the negative set. The radial basis (4) and polynomial kernel (5) (degree ranged from 1 to 6) functions were used for the SVMs. Since the polynomial kernels did not generate good results, we only present the results obtained from the radial basis kernel.

As the sizes of the positive and negative sets are substantially different, the performance of SVMs was evaluated for precision, defined as $tp/(tp + fp)$; and recall, defined as $tp/(tp + fn)$, where tp, tn, fp, and fn are the numbers of predicted true positive, true negative, false positive and false negative, respectively. In addition, the F-score combining the precision and recall:

$$F\text{-score} = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}, \tag{6}$$

was also evaluated. The reported values of precision, recall, and F-score are the averages from the 5-fold CVs.

The generalization performance of an SVM is controlled by the following parameters:

(1) C : the trade-off between the training error and the class separation;
(2) γ : the parameter in the radial basis function $\exp(-\gamma\|D_k\boldsymbol{x}_k^i - D\boldsymbol{x}_k^j\|^2)$;
(3) J : the biased penalty for errors from positive and negative training points.

The penalty term $C\sum_{i=1}^{m}\xi_i$ in SVM is split into two terms [22]:

$$C\sum_{i=1}^{m}\xi_i \Rightarrow C\sum_{\{i:y_i=1\}}\xi_i + CJ\sum_{\{i:y_i=-1\}}\xi_i. \tag{7}$$

The choices of the parameters in this experiment are given as follows:
for the new kernels,

C: from 1 to 40 with an incremental size of 3;
γ: from 0.001 to 1 with an incremental size of 0.003;
J: from 0.1 to 3.0 with an incremental size of 0.4;

and for the conventional k-peptides compositions,

C: from 1 to 150 with an incremental size of 10;
γ: from 1 to 100 with an incremental size of 10;
J: from 0.1 to 3.0 with an incremental size of 0.2.

The SVMLight package was used as the SVM solver [14]. The values of precision and recall of a 5-fold CV were computed for each triplet (C, γ, J). The best values of precision, recall and the corresponding F-score for each method are reported. The symbols P, R and F used in Tables 1 and 2 stand respectively for precision, recall, and F-score.

From Table 1, it can be seen that the performance is sensitive to the value of k. With $k = 2$, the new kernel achieves the best performance in terms of precision, recall, and F-score. Specifically, the recall (85.73) is about 10% higher compared with that (75.76) obtained when $k = 3$, while maintaining a similar level of precision; the precision (90.07) is about 8% higher than that (81.93) obtained when $k = 1$; while keeping almost the same recall value.

The results of prediction with the conventional k-peptide composition scheme for the same data set are reported in Table 2. It is readily seen from the table that the three coding methods do not show significant difference in their performance, although the coding with composition ($k = 1$) achieves a slightly better level

Table 1. Results obtained from the new kernel method with different matrices for the proteins from Gram-negative bacteria

Method	D1			D2			D3		
Localization	P	R	F	P	R	F	P	R	F
Cytoplasmic	76.74	87.05	81.46	88.12	84.53	86.24	77.38	73.48	75.38
Inner membrane	95.30	84.95	89.69	95.39	90.73	92.90	97.29	85.27	90.88
Periplasmic	76.43	79.69	77.88	80.44	82.55	81.36	85.98	68.45	76.22
Outer membrane	84.92	90.72	87.63	95.20	92.83	93.95	96.25	86.73	91.24
Extra cellular	76.26	83.73	79.73	91.22	78.00	83.85	92.11	64.86	76.12
Average	**81.93**	**85.23**	**83.28**	**90.07**	**85.73**	**87.66**	**89.80**	**75.76**	**81.94**

Table 2. Results obtained from the conventional k-peptide coding method for the proteins from Gram-negative bacteria

Method	composition			di-peptide			tri-peptide		
Localization	P	R	F	P	R	F	P	R	F
Cytoplasmic	80.09	70.77	74.66	81.12	57.69	66.09	83.43	45.00	55.09
Inner membrane	98.52	82.27	89.54	98.15	81.51	88.80	99.52	80.75	89.01
Periplasmic	94.12	55.17	68.38	91.80	54.14	65.77	90.37	50.34	63.11
Outer membrane	87.86	84.23	85.74	90.12	79.76	84.00	93.15	83.29	87.79
Extra cellular	88.38	53.68	66.05	89.71	53.68	66.27	92.57	50.53	64.63
Average	**89.79**	**69.23**	**76.87**	**90.18**	**65.36**	**74.18**	**93.17**	**64.80**	**74.62**

of recall. In this comparison it is clear the new kernel method demonstrates superior performance over the conventional k-peptide coding method. The recall (85.73) produced by the new method with $k = 2$ shows substantial improvement from 69.23 (composition), 65.36 (di-peptide), and 64.80 (tri-peptide); the F-score is likewise improved to a level of 87.66, from 76.87 (composition), 74.18 (di-peptide), and 74.62 (tri-peptide); while a similar level of precision is maintained.

The performance of the new kernel method also compares favorably with SCL-BLAST [23], a BLAST-search based predictor for all localizations. The new method improves recall from 60.40 to 85.73 and F-score from 74.36 to 87.66, while having a lower precision (90.07) compared to that (96.70) of SCL-BLAST.

It is worth noting that the new method ($k = 2$) yields a similar overall performance compared with the latest version of PSORT-B (v.2.0) [9], which gives a precision of 95.88, a recall of 82.6 and an F-score of 88.7. As the PSORT-B comprises several modules designed for the prediction of specific localization sites, it is surprising that our single module can match the performance of this integrative predictor.

4 Discussion

Kernel-based learning algorithms, such as SVMs, are among the most advanced machine learning methods. The success largely depends on the choice of kernel functions. In general, the more that prior knowledges is incorporated into the kernel function, the better the performance of the SVMs. Several successful approaches have focused on the design of new kernels reflecting higher levels of biological knowledge. This includes the mismatch kernel for protein fold recognition [17], the Fisher kernel for the detection of remote protein homologies [13], a class of edit kernels for the prediction of translation initiation sites in eukaryotic mRNAs [18], and an oligo kernel for the prediction of prokaryotic translation initiation sites [20]. The approach most relevant to our study is the mismatch kernel. In that work, each protein sequence is coded by a vector with each entry representing the number of occurrences of a k-peptide including its mismatched partners, namely, those that have a limited number of mutated amino acids in reference to the original k-peptide. Then, a linear kernel is essentially a weighted sum of numbers of shared mismatched k-peptides between two sequences. The class of new kernels proposed in this study can be considered as a generalization

of the mismatch kernel. The similarity between two k-peptides is measured not only by the number of mismatched residues, but also by the evolution distances between the residues based on their BLOSUM scores. This is concluded that these features are the basis of the improved performance of the new kernels that is revealed in the comparison with the conventional k-peptide coding scheme.

Although the class of the new kernels proposed in this study is general for any k-peptides, the implementation presents a particular difficulty when $k > 3$. This is why the experiments in this work were performed with $k \leq 3$. A clever data structure, such as the one used in [17], is needed for fast computation. This issue is currently under investigation.

5 Conclusions

This work has introduced a class of novel kernels based on matrices formed by the BLOSUM scores assigned to pairs of k-peptides of protein sequences. Through a linear mapping defined by the matrix, this method generalized the conventional k-peptide coding method to allow the measurement of similarity between mismatched k-peptides based on BLOSUM scores. The kernels have been used in support vector machines for the prediction of subcellular localizations. The performance of the new kernels was evaluated on a set of proteins with experimentally determined localizations from Gram-negative bacteria. Compared with other coding systems using k-peptide compositions, the experimental results demonstrate that the new kernel exhibited superior overall performance for the predictions. The method also achieved a similar level of overall performance comparing with that of the integrated system PSORT-B.

Acknowledgments

This research is partially supported by National Science Foundation (EIA-022-0301) and Naval Research Laboratory (N00173-03-1-G016). The authors are thankful for Deepa Vijayraghavan for the assistant with computing environment.

References

1. Bannai, H., Tamada, Y., Maruyama, O., Nakai, K., Miyano, S.: Extensive feature detection of N-terminal protein sorting signals. Bioinformatics **18** (2002) 298-305
2. Cai, Y.D., Chou, K.C.: Predicting subcellular localization of proteins in a hybridization space. Bioinformatics **20** (2003) 1151-1156
3. Chou, K.C., Cai, Y.D.: Using functional domain composition and support vector machines for prediction of protein subcellular location. J. Biol. Chem. **277** (2002) 45765-4576
4. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines, Cambridge University Press (2000)
5. Emanuelsson, O., Nielsen, H., Brunak, S., von Heijne, G.: Predicting subcellular localization of proteins based on their N-terminal amino acid sequence. J. Mol. Biol. **300** (2000) 1005-1016

6. Emanuelsson, O.: Predicting protein subcellular localisation from amino acid sequence information. Brief. Bioinform. **3** (2002) 361-376
7. Feng, Z.P.: Prediction of the subcellular location of prokaryotic proteins based on a new representation of the amino acid composition. Biopolymers **58** (2001) 491–99
8. Gardy, J.L. *et al.*: PSORT-B: improving protein subcellular localization prediction for Gram-negative bacteria. Nucleic Acids Res. **31** (2003) 3613-3617
9. Gardy, J.L. *et al.*: PSORTb v.2.0: expanded prediction of bacterial protein subcellular localization and insights gained from comparative proteome analysis. Bioinformatics **21** (2005) 617-623
10. von Heijne, G.: Signals for protein targeting into and across membranes. Subcell. Biochem. **22** (1994) 1-19
11. Horton, P., Nakai, K.: PSORT: a program for detecting sorting signals in proteins and predicting their subcellular localization. Trends Biochem. Sci. **24** (1999) 34-36
12. Hua, S., Sun, Z.: Support vector machine approach for protein subcellular localization prediction. Bioinformatics **17** (2001) 721-728
13. Jaakkola, T., Diekhans, M., Haussler, D.: Using the Fisher kernel method to detect remote protein homologies. Proc. of the Seventh International Conference on Intelligent Systems for Molecular Biology (1999) 149 - 158
14. Joachims, T.: Making Large Scale SVM Learning Practical. Advances in Kernel Methods-Support Vector Learning. MIT Press, Cambridge (1999)
15. Lei, Z, Dai, Y.: A novel approach for prediction of protein subcellular localization from sequence using Fourier analysis and support vector machines. Proc. of the Fourth ACM SIGKDD Workshop on Data Mining in Bioinformatics (2004) 11-17
16. Lei, Z, Dai, Y.: A new kernel based on high-scored pairs of tri-peptides and its application in prediction of protein subcellular localization. Proc. of International Conference on Computational Science (ICCS 2005), LNCS **3515** (2005) 903-910
17. Leslie, C., Eskin, E., Cohen, A., Weston, J., Noble, W.: Mismatch string kernels for discriminative protein classification. Bioinformatics **20** (2004) 467-476
18. Li, H., Jiang, T.: A class of edit kernels for SVMs to predict translation initiation sites in eukaryotic mRNAs. Proc. of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB) (2004) 262-271
19. Lu, Z., Szafron, D., Greiner, R., Lu, P., Wishart, D.S, Poulin, B., Anvik, J., Macdonell, C., Eisner, R.: Predicting subcellular localization of proteins using machine-learned classifiers. Bioinformatics **20** (2004) 547-556
20. Meinicke, P., Tech, M., Morgenstern, B., Merkl, R.: Oligo kernels for datamining on biological sequences: a case study on prokaryotic translation initiation sites. BMC Bioinformatics **5** (2004) 169
21. Menne, K. M. L., Hermjakob, H., Apweiler, R.: A comparison of signal sequence prediction methods using a test set of signal peptides. Bioinformatics **16** (2000) 741–742
22. Morik, K., Brockhausen, P,. Joachims, T.: Combining statistical learning with a knowledge-based approach - A case study in intensive care monitoring. Proc. of the Sixteenth International Conference on Machine Learning (1999) 268-277
23. Nair, R., Rost, B.: Sequence conserved for subcellular localization. Protein Sci. **11** (2002) 2836-2847
24. Nakai, K.: Protein sorting signals and prediction of subcellular localization. Adv. Protein. Chem. **54** (2000) 277-344
25. Nakai, K., Kanehisa, M.: Expert system for predicting protein localization sites in Gram-negative bacteria. Proteins **11** (1991) 95-110

26. Nielsen, H., Engelbrecht, J., Brunak, S., von Heijne, G.: A neural network method for identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. Int. J. Neural Syst. **8** (1997) 581-599
27. Park, K., Kanehisa, M.: Prediction of protein subcellular locations by support vector machines using compositions of amino acids and amino acid pairs. Bioinformatics **19** (2003) 1656-1663
28. Reinhardt, A., Hubbard, T.: Using neural networks for prediction of the subcellular location of proteins. Nucleic Acids Res. **26** (1998) 2230–2236
29. Tusnady, G.E., Simon, I.: Principles governing amino acid composition of integral membrane proteins: application to topology prediction. J. Mol. Biol. **283** (1998) 489–506
30. Tusnady, G.E., Simon, I.: The HMMTOP transmembrane topology prediction server. Bioinformatics **17** (2001) 849–850
31. Yu, C.S., Lin, C.J., Hwang, J.K.: Predicting subcellular localization of proteins for Gram-negative bacteria by support vector machines based on n-peptide compositions. Protein Sci. **13** (2004) 1402–1406

A Protein Structural Alphabet and Its Substitution Matrix CLESUM

Wei-Mou Zheng[1,⋆] and Xin Liu[2]

[1] Institute of Theoretical Physics, Academia Sinica, Beijing 100080, China
[2] The Interdisciplinary Center of Theoretical Studies, Academia Sinica Beijing 100080, China

Abstract. By using a mixture model for the density distribution of the three pseudobond angles formed by C_α atoms of four consecutive residues, the local structural states are discretized as 17 conformational letters of a protein structural alphabet. This coarse-graining procedure converts a 3D structure to a 1D code sequence. A substitution matrix between these letters is constructed based on the structural alignments of the FSSP database. [1]

1 Introduction

Drastic approximations are unavoidable in prediction of protein structure from the amino acid sequence. Generally, the procedure to deduce finite discrete conformational states from a continuous conformational phase space is a clustering analysis. There have been a variety of different ways of clustering. For example, Park and Levitt (1995) represent the polypeptide chain by a sequence of rigid fragments that are chosen from a library of representative fragments, and concatenated without any degrees of freedom. The average deviation of the global-fit approximations over a training set is taken as the objective function for optimizing the finite representative fragments. The state clusters there are representative points of the phase space. Rooman, Kocher and Wodak (1991) intuitively divide the ϕ-ψ space into 6 regions, which corresponds to a partitioning based on the Ramachandran plot. Standard methods for clustering analysis have been also used to generate discrete structure states (Bystroff and Baker, 1998).

Hidden Markov models (HMMs; Rabiner, 1989), possessing a rigorous but flexible mathematical structure, have been used in a variety of computational biology problems such as sequence motif recognition (Fujiwara et al., 1994), gene finding (Burge and Karlin, 1997), protein secondary structure prediction (Asai, Hazamizu and Handa, 1993; Zheng, 2004), and multiple sequence alignments (Krogh *et al.*, 1994). The HMMs have been also used for identifying the modular framwork for the protein backbone (Edgoose, Allison and Dowe, 1998; Camproux *et al.*, 1999). In these HMMs conformation states are represented by probability

⋆ Presenter, to whom correspondence should be addressed zheng@itp.ac.cn.
[1] This work was supported in part by the Special Funds for Major National Basic Research Project and the National Natural Science Foundation of China.

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 59–67, 2005.

distributions, which is much finer than a simple partition of the phase space. HMMs involve in a large number of parameters, and it is not so convenient to assign structure codes to a short segment with HMMs. Here we develop a description of protein backbone tertiary structure using psuedobond angles of successive C_α atoms. Finite conformational states as structural alphabet are selected according to the density peaks of probability distribution in the phase space spanned by pseudobond angles. We derive a substitution matrix of these states from a representative pairwise aligned structure set of the FSSP (families of structurally similar proteins) database of Holm and Sander (1994).

2 Methods

Pseudobond Angles. Among a variety of abstract representing forms for protein 3D structure, a frequently encountered one is the protein virtual backbone forming by C_α atoms. The virtual bond bending angle θ defined for three contiguous points (a, b, c) is the angle between the vectors $\mathbf{r}_{ab} = \mathbf{r}_b - \mathbf{r}_a$ and $\mathbf{r}_{bc}$, i.e. $\theta = \mathbf{r}_{ab} \cdot \mathbf{r}_{bc}/(|r_{ab} r_{bc}|)$. The range of θ is $[0, 2\pi]$. The virtual bond torsion angle τ defined for four contiguous points (a, b, c, d) is the dihedral angle between the planes *abc* and *bcd*. The range of τ is $(-\pi, \pi]$, and its sign is the same as $(\mathbf{r}_{ab} \times \mathbf{r}_{bc}) \cdot \mathbf{r}_{cd}$. In fact, we may adopt a wider range of τ under the equivalence relation that τ_1 and τ_2 are equivalent if $\tau_1 = \tau_2 \pmod{2\pi}$. For the four-residue segment *abcd*, by takeing a as the origin, and b on the x-axis, and c on the xy-plane, the number of independent relative coordinates are 6. The assumption of the fixed pseudobond length, which is 3.8 Å for the dominating *trans* peptide, further reduces the number of degrees of freedom to 3. These independent coordinates correspond to the angles $(\theta_{abc}, \tau_{abcd}, \theta_{bcd})$. Elongating the segment by one residue e will add two more angles τ_{bcde} and θ_{cde}. Generally, for a sequence of n residues, we have $n-2$ bending angles and $n-3$ torsion angles, $2n-5$ in total. We shall assign the angles $(\theta_{abc}, \tau_{abcd}, \theta_{bcd}) \equiv (\theta_b, \tau_c, \theta_c)$ to residue c, the third of the four-residue segment.

By convention, for the chain $\{\mathbf{r}_0, \mathbf{r}_1, \cdots, \mathbf{r}_n\}$ with angles $\{\theta_1; \tau_2, \theta_2; \cdots; \tau_{n-1}, \theta_{n-1}\}$, we set the origin at $\mathbf{r}_0$, put $\mathbf{r}_1$ along the x-axis, and add $\tau_1 = 0$. Introducing the identity matrix I, the rotation matrices R_θ and R_τ

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \; R_\tau = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\tau & -\sin\tau \\ 0 & \sin\tau & \cos\tau \end{pmatrix}, \text{ and } \mathbf{d} = \mathbf{r}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad (1)$$

position $\mathbf{r}_k$ is determined by

$$T_0 = I, \; \mathbf{r}_0 = 0 \cdot \mathbf{d}, \; T_k = T_{k-1} R_{\tau_k} R_{\theta_k}, \; \mathbf{d}_k = T_{k-1} \cdot \mathbf{d}, \; \mathbf{r}_k = \mathbf{r}_{k-1} + \mathbf{d}_k, \; k \geq 1. \quad (2)$$

Longer fragments will include more correlation than shorter fragments. However, the complexity that can be explored with the longer fragment lengths is limited severely by the relatively small number of known protein structures, and a larger number of discrete states have to be determined for a longer segment.

The minimal unit where the relative coordinates fix the angles and vice versa is four contiguous residue segment. We shall concentrate mainly on the structure codes for the four residue unit.

The Mixture Model for the Angle Probability Distribution. The three pseudobond angles (θ, τ, θ') of the four-residue unit span the three-dimensional phase space. Our classifiers for conformational states are based on the following mixture model M: The probability distribution of 'points' $\mathbf{x} \equiv (\theta, \tau, \theta')$ is given by the mixture of several normal distributions

$$P(\mathbf{x}|M) = \sum_{i=1}^{c} \pi_i N(\mu_i, \mathbf{\Sigma}_i), \tag{3}$$

where c is the number of the normal distribution categories in the mixture, π_i the prior for category i, and $N(\mu, \mathbf{\Sigma})$ the normal distribution. These categories will be translated as the structure codes.

To objectively determine the number c of categories, we investigate density peaks in the phase space with the downhill simplex method of Nelder and Mead (1965). We use counts in a rectangular box as the value of the function for optimization at the center of the box. We examine also density peaks in the five-dimensional phase space spanned by $(\theta_b, \tau_c, \theta_c, \tau_d, \theta_d)$ of the five-residue unit *abcde*. It is demanded that all the important three-angle modes implied by the main density peaks in the five-angle phase space must be included in the modes used for the construction of the mixture model.

The main purpose of searching for density peaks is to estimate the number c of categories and $\{\mu_i\}$ for each category. Once this has been done, we may start with some simple $\{\pi_i\}$ and $\{\mathbf{\Sigma}_i\}$, say $\pi_i = 1/c$ and certain diagonal $\{\mathbf{\Sigma}_i\}$, and then update the mixture model by the Expectation-Maximization (EM) method. For each point $\mathbf{x}_k = (\theta_{k-1}, \tau_k, \theta_k)$, we calculate the probability for the point to belong to the i-th category C_i according to the Bayes formula as

$$P(C_i|\mathbf{x}_k) \propto \pi_i P(\mathbf{x}_k|C_i) \propto \pi_i |\mathbf{\Sigma}_i|^{-1/2} \exp[\tfrac{1}{2}(\mathbf{x}_k - \mu_i) \cdot \mathbf{\Sigma}_i^{-1} \cdot (\mathbf{x}_k - \mu_k)], \tag{4}$$

where we always shift τ_k to the interval $[\tau^{(i)} - \pi, \tau^{(i)} + \pi)$ centered at $\tau^{(i)}$ of the τ-component of the mean μ_i. Generally, the objective function for optimizing the mixture model is

$$\text{Prob}(\{\mathbf{x}_k\}) = \prod_k \sum_i P(\mathbf{x}_k, C_i) \propto \prod_k \sum_i P(C_i|\mathbf{x}_k). \tag{5}$$

However, when we convert point $\mathbf{x}_k$ to its structural code i^*, we use

$$i^* = \arg_i \max P(C_i|\mathbf{x}_k). \tag{6}$$

An alternative objective function would be $Q(\{\mathbf{x}_k\}) = \prod_k \max_i P(C_i|\mathbf{x}_k)$. When starting with narrow distributions for $\mathbf{\Sigma}_i$, a very high value of Q could be seen at the first step. However, by just one step of the EM iteration Q will drop significantly, and then increases at later steps. While $\text{Prob}(\{\mathbf{x}_k\})$ never decreases,

Q will decrease after reaching its maximum. We may stop the model training before Q decreases again. Thus, the optimization here is a compromise between $\text{Prob}(\{\mathbf{x}_k\})$ and $Q(\{\mathbf{x}_k\})$. Once we have the model, we may convert a structure to its conformational code sequence according to (6).

3 Result

For establishing the discrete structural states by training the mixture model, we create a nonredundant set of 1544 non-membrane proteins from PDB_SELECT with amino acid identity less than 25% issued on 25 September of 2001. The data of the three-dimensional structures for these proteins are taken from Protein Data Bank (PDB). The total number of contiguous fragments is 2248, which gives totally 264,232 points in the three-angle phase space.

The Discrete Structural States. The marginal one-dimensional distribution of the pseudobond bending angle has two prominent peaks around $\theta = 1.10$ and 1.55 (radians). Non-zero θs are in the interval $[.4, 1.9]$. The marginal one-dimensional distribution of the torsion angle τ has one immediately noticeable peak at $\tau = 0.87$ (corresponding to the helix). Another peak at $\tau = -2.94$ is less prominent. There is a vague peak still recognizable around $\tau = -2.00$. A grid generated with $\theta \in \{1.00, 1.55\}$ and $\tau \in \{-2.80, -2.05, -1.00, 0.00, 0.87\}$ is used to search high dimensional phase space for density peaks by the downhill simplex method. In the box counting, the box size is taken from 0.1 to 0.2 for θ, and the width for τ is twice of that for θ. Further exploring main peaks in the

Table 1. The 17 structural states from the mixture model

State	π	$\lvert\Sigma\rvert^{-1/2}$	μ			$\mathbf{\Sigma}^{-1}$					
			θ	τ	θ'	$\theta\theta$	$\tau\theta$	$\tau\tau$	$\theta'\theta$	$\theta'\tau$	$\theta'\theta'$
I	8.2	1881	1.52	0.83	1.52	275.4	−28.3	84.3	106.9	−46.1	214.4
J	7.3	1797	1.58	1.05	1.55	314.3	−10.3	46.0	37.8	−70.0	332.8
H	16.2	10425	1.55	0.88	1.55	706.6	−93.9	245.5	128.9	−171.8	786.1
K	5.9	254	1.48	0.70	1.43	73.8	−13.7	21.5	15.5	−25.3	75.7
F	4.9	105	1.09	−2.72	0.91	24.1	1.9	10.9	−11.2	−8.8	53.0
E	11.6	109	1.02	−2.98	0.95	34.3	4.2	15.2	−9.3	−22.5	56.8
C	7.5	100	1.01	−1.88	1.14	28.0	4.1	6.2	2.3	−5.1	69.4
D	5.4	78	0.79	−2.30	1.03	56.2	3.8	4.2	−10.8	−2.1	30.1
A	4.3	203	1.02	−2.00	1.55	30.5	9.1	8.7	6.0	5.7	228.6
B	3.9	66	1.06	−2.94	1.34	26.9	4.6	4.9	9.5	−5.0	54.3
G	5.6	133	1.49	2.09	1.05	163.9	0.6	3.8	2.0	−3.7	32.3
L	5.3	40	1.40	0.75	0.84	43.7	2.5	1.4	−7.0	−2.9	34.5
M	3.7	144	1.47	1.64	1.44	72.9	2.1	4.8	1.9	−7.9	72.9
N	3.1	74	1.12	0.14	1.49	25.3	3.2	3.1	9.9	0.9	83.0
O	2.1	247	1.54	−1.89	1.48	170.8	−0.7	3.7	−4.1	3.1	98.7
P	3.2	206	1.24	−2.98	1.49	48.0	8.2	7.3	−4.9	−6.6	155.6
Q	1.7	25	0.86	−0.37	1.01	28.4	1.5	1.2	3.4	0.1	19.5

five-angle phase space, we identify 17 mode centers, which are then used as the main initial parameters to train the mixture model. Finally, the 17 structural states are obtained for the mixture model by the EM algorithm. They are listed in Table 1.

The total number of possible four-residue secondary structures is 37 due to the restriction of the minimal lengths 2 for e and 3 for h. It is seen that there exists a correlation between the 17 structural states and the secondary structures. For example, *hhhh* are mainly attributed to H, I and J, while *eeee* to E and D. The mutual information between the conformational codes and the secondary structure states equals 0.731. Conformation *cccc* has rather uniform percentages in different structural states as we would expect.

Structural Substitution Matrix. Amino acid substitution matrices, extracted from our knowledge of most and least common changes in a large number of proteins, serve for the purpose of sequence alignment. The popular BLOSUM matrix of Henikoff and Henikoff (1992) is derived from a large set of conserved amino acid patterns without gaps representing various families. The frequency of amino acid substitutions in alignments is counted in sequence alignments. These frequencies are then divided by the expected frequency of finding the amino acids together in an alignment by chance. The ratio of the observed to the expected counts is an odds score. The BLOSUM entries are logarithms of the odds scores with the base 2 and multiplied by a scaling factor of 2.

To use our structural codes directly for the structural comparison, a score matrix similar to BLOSUM is desired. There is a database of aligned structures, the FSSP of Holm and Sander (1997), which is based on exhaustive all-against-all 3D structure comparison of protein structures in the PDB. The proteins

Table 2. CLESUM: The conformation letter substitution

	J	H	I	K	N	Q	L	G	M	B	P	A	O	C	E	F	D
J	38																
H	15	25															
I	12	14	51														
K	16	8	17	51													
N	−1	−32	−16	28	89												
Q	−43	−87	−69	−24	31	88											
L	−31	−61	−48	0	5	24	71										
G	-21	−49	−40	−11	−7	8	27	68									
M	17	−2	−4	14	8	−7	4	21	59								
B	−55	−94	−79	−49	−11	10	−13	12	−14	49							
P	−33	−58	−55	−35	−4	6	−14	3	7	41	64						
A	−22	−43	−39	−17	10	13	−12	−7	−2	19	34	71					
O	−23	−54	−37	5	14	−13	−5	−2	5	−12	2	23	102				
C	−42	−75	−59	−32	−5	27	−2	−6	−12	5	4	12	1	51			
E	−91	−125	−112	−83	−43	−8	−23	−24	−47	13	−6	−27	−49	2	34		
F	−73	−106	−95	−67	−32	0	−18	−6	−34	4	−2	−22	−31	19	24	48	
D	−87	−122	−105	−81	−45	13	−24	−32	−50	11	−11	−19	−43	19	21	20	49

in the FSSP are divided into a representative set and sequence homologs of the representative set. The representative set contains no pair which have more than 25% sequence identity. Family indices of the FSSP are obtained by cutting the tree at levels of 2, 4, 8, 16, 32 and 64 standard deviations above database average. We convert the structures of the representative set to their structural code sequences. All the pair alignments of the FSSP (version of Oct 2001) for the proteins with the same first three family indices in the representative set are collected for counting aligned pairs of structural codes. The total number of code pairs are 1,143,911. The substitution matrix derived in the same way as the BLOSUM was obtained (without sequence clustering) is shown in Table 2, where a scaling factor of 20 instead of 2 is used to show more details. We call this conformation letter substitution matrix CLESUM. Henikoff and Henikoff (1992) introduced for their BLOSUM the average mutual information per amino acid pair H, which is the Kullback-Leibler distance between the joint model of the alignment and the independent model. The value of H for our CLESUM equals 1.05, which is close to that for BLOSUM83.

4 Discussion

Biologically important modules have been repeatedly employed in protein evolution by gene duplication and rearrangement mechanisms. They form components of fundamental units of structure and function. The presence of modules provides a guide to classify proteins into module-based families, and helps the structure prediction. The existence of such conservative recurrent segments sets a solid foundation for the local analysis. The parameter number of HMM increases quadratically with the number of categories, while only linearly for a mixture model. We have to compromise between precision and correlation. A mixture model with fine categories is also promising. We have discretized the combination of three psuedobond angles formed by four consecutive C_α atoms to convert the local geometry to 17 coarse-grained conformational letters according to a mixture model of the angle distribution.

The Precision of the Conformational Codes. From the correlation between the conformational codes and the secondary structures, it is not surprising that there exists a propensity of the codes to amino acids. The coarse-graining would introduce an error. It is then important to examine the precision of the codes. For this purpose, we randomly pick up 1,000 points for each code, and calculate the distance root mean squared deviation (*drms*) for each of the total 499,500 pairs from their coordinates. The *drms* of structures a and b is defined without requiring a structure alignment as the averaged distance pair difference

$$drms = \left[\frac{2}{n(n-1)} \sum_{i=2}^{n} \sum_{j=1}^{i-1} (|\mathbf{r}_{ai} - \mathbf{r}_{aj}| - |\mathbf{r}_{bi} - \mathbf{r}_{bj}|)^2 \right]^{1/2}, \tag{7}$$

where $\mathbf{r}_{ai}$ is the coordinate of atom i in structure a. The averaged coordinate pair difference, i.e. the coordinate root mean squared deviation *crms*, is about

1.2 times of the *drms*. The most precise code H has an error 0.133 ± 0.060Å,while the vaguest code L has an error 0.604 ± 0.365Å. After averaging over the code relative frequencies, the mean error is 0.330Å.

Structure Alignment via Conformational Codes. The conversion of a 3D structure of coordinates to its conformational codes requires little computation. To distinguish from the amino acid sequence, we call the converted code sequence the code series, or simply series. Once we transform 3D structures to 1D series, the structure comparison becomes the series comparison. Tools for analyzing ordinary sequences can be directly applied. We have constructed the conformational letter substitution matrix CLESUM from the alignments of the FSSP database. We shall examine the performance of the conformational alphabet derived above.

```
1urnA avpetRPNHTIYINNLNEKIKKDELKKSLHAIFSRFGQILDILVSRS
1ha1b    ahLTVKKIFVGGIKEDT    EEHHLRDYFEQYGKIEVIEIMTDRGS
        CCPMCEALEEEENGCPJGCCIHHHHHHHHHIKMJILQEPLDEEEBGAIK
        ...BBEBGEDEENMFNMLFA....HHHHHKKMJJLCEBLDEBCECAKK

1urnA LKMRGQAFVIFKEVSSATNALRSMqGFPFYDKPMRIQYAKTDSDIIAKM
1ha1b GKKRGFAFVTFDDHDSVDKIVIQ kYHTVNGHNCEVRKAL
      ...GNGEDBEEALAJHHHHHHHIKKGNGCENOGCCEFECCALCCAHIJH
      AGCPOLEDEEEALBJHHHHI.IJGALEEENOGBFDEECC.........
```

Fig. 1. The alignment of 1urnA and 1ha1. The first two lines are their amino acid sequences aligned according to the FSSP, while the last two lines are the global Needleman-Wunsch alignment of the conformational code series. Lowercase letters of amino acids indicate structural nonequivalence.

Holm and Sander (1998) gave an example of the α/β-meander cluster with four members showing different levels of structural similarity. Their PDB-IDs are 1urnA, 1ha1, 2bopA and 1mli. The structure of 1urnA was taken as the frame to superimpose the other structures. The structural similarity to 1urnA from high to low are 1ha1, 2bopA and 1mli. Taking the scaling factor for the CLESUM to be 2, and using -12 for the the gap-opening penalty and -4 for the gap extension, the global Needleman-Wunsch alignment of 1urnA and 1ha1 is shown in Fig. 1, where, in the first two lines, the amino acid sequences aligned according to the FSSP are also given. It is seen that, except for segment boundaries, the two alignments coincide. The alignment of the FSSP and the code series alignment for 1urnA and 2bopA have three common segments falling in positive score regions of the series alignment. In the alignments for 1urnA and 1mli two common segments longer than 8 are still seen. As for the amino acid sequence alignment, in the case of 1urnA and 1ha1 two segments of lengths 13 and 21 of the sequence alignment coincide with the FSSP, but no coincidence is seen in the other two cases.

The conformational codes are local. Even though a global alignment algorithm is used, this does not guarantee that the found alignment corresponds to the optimal structure superposition. However, the code series alignment does not affected by the domain move, is then good for analyzing the structure evolution. For example, the first helix of 1ha1 is shorter than its counterpart in 1urnA by one turn. The FSSP aligns the N-cap (with codes FA) of the 1ha1 helix to the helix (with codes HH) of 1urnA, but local structure FA is closer to CC (with positive scores) than to HH (with negative scores).

It is known that the sequence-structure relationships have not always been strong. Bystroff and Baker (1998) have built a library of structure-sequence motifs, which are expected to correspond to functional units recurring in different protein contexts and to be found in different combinations in distantly related or functionally unrelated proteins. To identify the structural features that have strong sequence preferences is to locate peaks of density distribution in the joint structure-sequence space. Previously, the structure-based clustering was a duty much heavier than the sequence-based clustering, so one had to start with a sequence-based clustering, and was kept constantly to run between the structure and sequence subspaces. It is then interesting to see whether the library can be improved by clustering directly in the joint structure-sequence space with the help of conformational codes. This is under study.

References

1. Asai,K., Hazamizu,S., and Handa, K. (1993): Secondary structure prediction by hidden Markov model, CABIOS **9**, 141-146.
2. Burge,C., and Karlin,S. (1997): Prediction of complete gene structures in human genomic DNA, J. Mol. Evol. **268**, 78-94.
3. Bystroff,C., and Baker,D. (1998): Prediction of local structure in proteins using a library of sequence-structure motifs J. Mol. Biol. **281**, 565-577.
4. Camproux,A.C., Tuffery,P., Chevrolat,J.P., Boisvieux,J.F., and Hazout,S. (1999): Hidden Markov model approach for identifying the modular framework of the protein backbone, Protein Eng. **12**, 1063-1073.
5. Edgoose,T., Allison,L. and Dowe,D.L. (1998): An MML classification of protein structure that knows about angles and sequences, pp585-596, Proc. 3rd Pacific Symposium on Biocomputing (PSB-98), Hawaii, USA.
6. Fujiwara,Y., Asogawa,M. and Konagaya,A. (1994): Stochastic motif extraction using hidden Markov model, Proc. ISMB94, pp.121-129.
7. Henikoff,S., and Henikoff,J.G. (1992): Amino acid substitution matrices from protein blocks, Proc. Natl. Acad. Sci. USA **89**, 10915-10919.
8. Holm,L., and Sander,C. (1998): Touring protein fold space with Dali/FSSP, Nucleic Acids Research **26**, 316-319.
9. Krogh,A., Brown,M., Mian,I.S., Sjölander,K., and Haussler,D. (1994): Hidden Markov models in computational biology: Applications to protein modeling, J. Mol. Biol. **235**, 1501-1531.
10. Nelder,J.A., and Mead,R. (1965): A simplex method for function minimization, Computer J. **7**, 308C313.
11. Park,B.H., and Levitt,M. (1995): The complexity and accuracy of discrete state models of protein structure, J. Mol. Biol. **249**, 493C507.

12. Rabiner,L.R. (1989): A tutorial on hidden Markov model and selected applications in speech recognition, Proc. IEEE **77**, 257-285.
13. Rooman,M.J., Kocher,J.-P.A., and Wodak,S.J. (1991). Prediction of protein backbone conformation based on seven structure assignments: Influnce of local interactions J. Mol. Biol. **221**, 961-979.
14. Zheng,W.M. (2004): Clustering of amino acids for protein secondary structure prediction, J. Bioinfor. Comp. Biol., **2** (2004) 333-342.

KXtractor: An Effective Biomedical Information Extraction Technique Based on Mixture Hidden Markov Models

Min Song, Il-Yeol Song, Xiaohua Hu, and Robert B. Allen

College of Information Science and Technology, Drexel University,
Philadelphia, PA 19104
(215) 895-2474, 01
{min.song, song, thu, rba}@drexel.edu

Abstract. We present a novel information extraction (IE) technique, KXtractor, which combines a text chunking technique and Mixture Hidden Markov Models (MiHMM). KXtractor overcomes the problem of the single Part-Of-Speech (POS) HMMs with modeling the rich representation of text where features overlap among state units such as word, line, sentence, and paragraph. KXtractor also resolves issues with the traditional HMMs for IE that operate only on the semi-structured data such as HTML documents and other text sources in which language grammar does not play a pivotal role. We compared KXtractor with three IE techniques: 1) RAPIER, an inductive learning-based machine learning system, 2) a Dictionary-based extraction system, and 3) single POS HMM. Our experiments showed that KXtractor outperforms these three IE systems in extracting protein-protein interactions. In our experiments, the F-measure for KXtractor was higher than for RAPIER, a dictionary-based system, and single POS HMM respectively by 16.89%, 16.28%, and 8.58%. In addition, both precision and recall of KXtractor are higher than those systems.

1 Introduction

The proliferation of the biomedical literature available on the Web is overwhelming. While the amount of data available to us is constantly increasing, our ability to absorb and process this information remains a challenging task. The biomedical literature has recently become a target domain that Information Extraction (IE) can be spotlighted on. IE scans text for information relevant to some interest, including extracting entities, relations, and events. In this paper, we propose a novel IE technique, called KXtractor, which employs Mixture Hidden Markov Models (MiHMMs) combined with a Support Vector Machine (SVM)-based text chunking technique.

MiHMM is defined as a mixture of Hidden Markov Models (HMMs) organized in a hierarchical structure to help the IE system cope with data sparseness. MiHMM takes a set of sentences with contextual cues that were identified by a Support Vector Machine-based text chunking technique. MiHMM then learns a generative probabilistic model of the underlying state transition structure of the sentence from a set of tagged training data. Given a trained probabilistic mixture model of the data,

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 68 – 81, 2005.

the system then applies this model to new unseen input documents to predict which portions of these documents are likely targets according to the training data template.

This paper investigates relationships between structure and performance of HMMs applied to information extraction problems. The sentence structure is diagnosed with POS taggers and an SVM-based text chunking technique. It is intuitive that different state configurations are appropriate for different types of extraction problems. What would be the effect of using the same structural template to train HMMs for different extraction tasks of varying levels of complexity? A simple structural template is used for HMM structure learning by the stochastic optimization algorithm in [6].

KXtractor is different from existing HMM-based approaches as follows: (a) It employs probabilistic mixture of HMMs that is hierarchically structured. (b) It incorporates contextual and semantic cues into the learned models to extract knowledge from the unstructured text collections without any document structures. (c) It adopts a SVM text chunking technique to partition sentences into grammatically related group. Thus using KXtractor for extracting biomedical entities has the following advantages over other approaches: (a) it overcomes the problem of the single POS HMMs with modeling the rich representation of text where features overlap among state units such as word, line, sentence, and paragraph. By incorporating sentence structures into the learned models, KXtractor provides better extraction accuracy than the single POS HMMs. (b) it resolves the issues with the single POS HMMs for IE that operate only on the semi-structured such as HTML documents and other text sources in which language grammar does not play a pivotal role.

With this novel and robust IE technique, we have extracted protein-protein pairs from abstracts in MEDLINE. We have compared the system performance of KXtractor with other IE techniques such as a rule-based learning, a dictionary-based, and single POS HMM techniques. Our experimental results show that KXtractor is superior to these techniques in most cases.

The rest of the paper is organized as follows: Section 2 summarizes the related work. Section 3 describes the overall architecture of KXtractor. Section 4 describes the evaluation. Section 5 reports on the experiments. Section 6 concludes the paper.

2 Related Works

Recently, there have been extensive studies on applying IE techniques to the biomedical literature. Much attention has been paid in extracting biomedical entities such as proteins or genes and their relations. Most of these studies adopt information extraction techniques, using a curated lexicon or natural language processing for identifying relevant tokens such as words or phrases in text [18].

In the area of named entity extraction, Fukuda et al. [8] extract protein names with hand-crafted rules. Although they reported that experimental results were competitive based on an F-value of 0.92, the results were not replicated and their method relied on manually created rules. Proux et al. [15] used single word names only with selected test set from 1200 sentences coming from Flybase. Collier et al. [4] adopted Hidden Markov Models (HMMs) for 10 test classes with small training and test sets. Krauthammer et al. [10] used a BLAST database with letters encoded as 4-tuples of DNA. Narayanaswamy et al. [14] used a Part of Speech (POS) tagger for tagging the

parsed MEDLINE abstracts. Although [14] and his colleagues implemented an automatic protein name detection system, the small number of words used made it difficult to demonstrate the usability of their system.

The second target object type of biomedical literature extraction is relation extraction. Leek [12] applied HMM techniques to identify gene names and chromosomes through heuristics. Blaschke et al. [1] extracted protein-protein interactions based on co-occurrence of the form "… p1…I1… p2" within a sentence, where p1, p2 are proteins and I1 is an interaction term. Protein names and interaction terms (e.g., activate, bind, inhibit) are provided as a "dictionary." Pustejovsky et al. [16] extracted an "inhibit" relation for the gene entity from MEDLINE. Jenssen et al. [9] extracted gene-gene relations based on co-occurrence of the form "… g1…g2…" within a MEDLINE abstracts, where g1 and g2 are gene names. Gene names were provided as a "dictionary", harvested from HUGO, LocusLink, and other sources. Although their study uses 13,712 named human genes and millions of MEDLINE abstracts, no extensive quantitative results are reported and analyzed.

Friedman et al. [7] extracted a pathway relation for various biological entities from a variety of articles. In their work, the precision of the experiments is high (from 79-96%). However, the recall was relatively low (from 21-72%). Bunescu et al. [2] conducted protein/protein interaction identification with several learning methods such as pattern matching rule induction (RAPIER), boosted wrapper induction (BWI), and extraction using longest common subsequences (ELCS). ELCS automatically learns rules for extracting protein interactions using a bottom-up approach. They conducted experiments in two ways; one with manually crafted protein names and the other with the extracted protein names by their name identification method. In both experiments, Bunescu et al. [2] compared their results with human-written rules and showed that machine learning methods provides higher precision than human-written rules.

KXtractor is differentiated from the previous approaches in that syntactical, as well as semantic cues, of input sentences are identified and incorporated into the extraction engines. By combining the text chunking technique and Mixture Hidden Markov Models, KXtractor takes advantage of sentence structures and patterns embedded in plain English sentences.

3 System Architecture

Figure 1 illustrates the system architecture of KXtractor. The system consists of two major components: 1) sentence chunking by SVM component and 2) relation extraction by the MiHMM component.

In the sentence chunking by SVM component, input data is plain text consisting of stitles and abstracts. The input data is separated into sentences. A set of regular expression rules are applied to parse sentences. For a parsed sentence, we applied an integrated POS tagging technique proposed by Song et al. [20] to tag sentences with POS. With SVM-based text chunking technique, these POS tagged sentences are then grouped into chunks of different phrase types such as noun, verb, and prepositions.

In the relation extraction by MiHMM component, MiHMM is applied to the grouped phrases by the SVM text chunking technique. The target state, which is a +target noun group containing proteins, is extracted with hierarchically structured HMMs. Finally protein-protein pairs are extracted from the target states within a sentence by re-applying MiHMM to the target state groups.

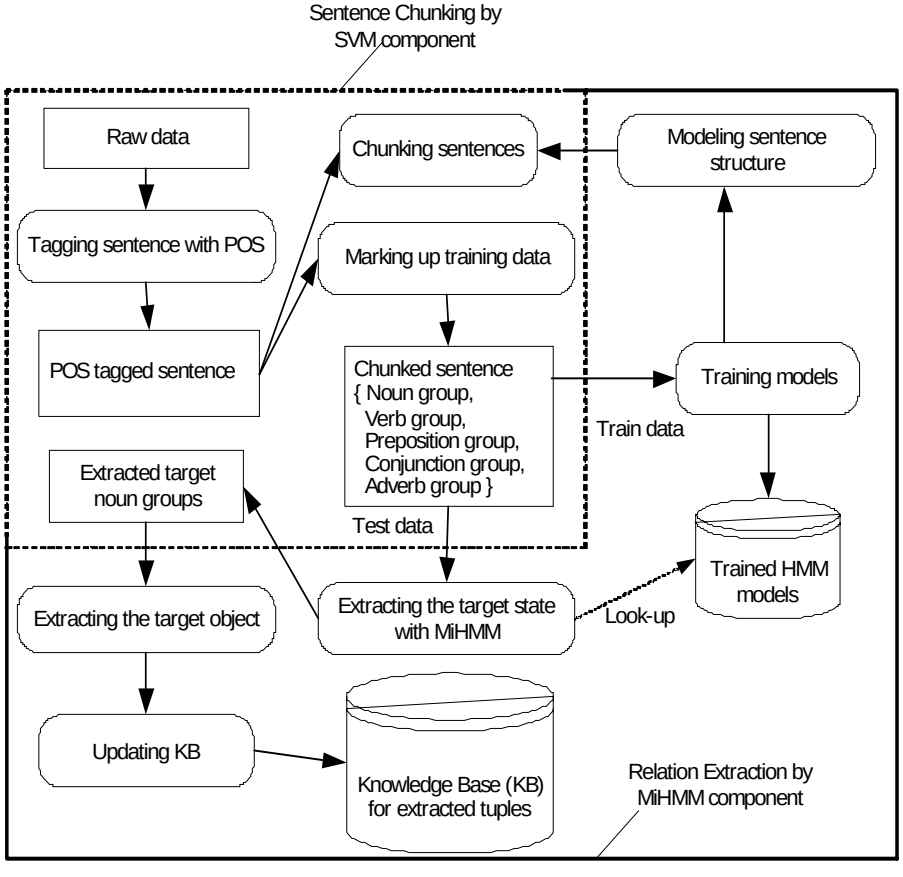
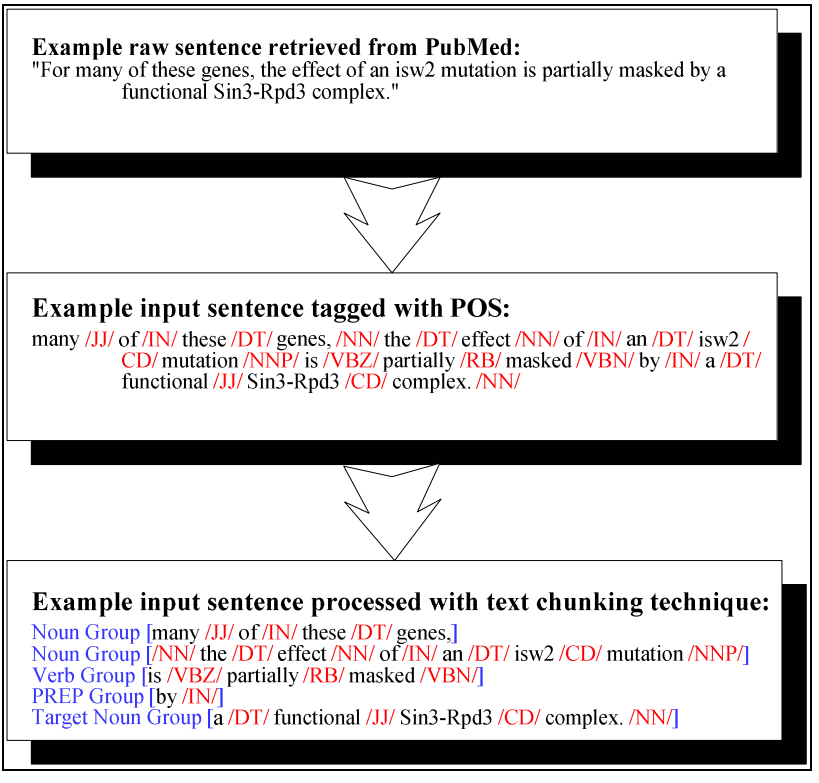


Fig. 1. System architecture of KXtractor

The results of running KXtractor are a set of tuples related to protein-protein pair. KXtractor stores these tuples in the knowledge base and resets the token statistics for the next input data. The detailed description of the components is provided in the sub-sections below.

Figure 2 illustrates the procedure of converting a raw sentence from PubMed to the phrase-based units grouped by the SVM text chunking technique. The top box shows a sentence that is part of an abstract retrieved from PubMed. The middle box illustrates



JJ denotes adjective, IN denotes preposition, DT denotes determiner, CD cardinal number, NN denotes singular noun, NNP denotes proper noun, VBZ denotes verb, VBN denotes verb, RB denotes adverb

Fig. 2. A procedure of sentence parsing

the parsed sentence by POS taggers. The bottom box shows the final conversion made to the POS tagged sentence by the SVM based text chunking technique.

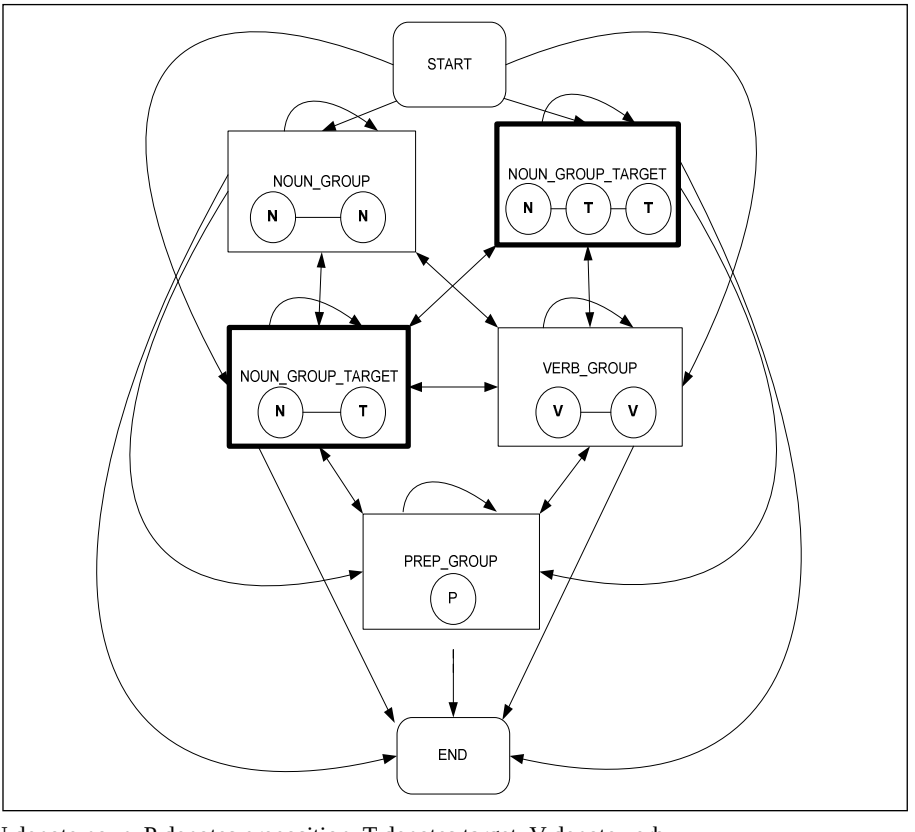
3.1 Sentence Chunking by SVM Component

Text chunking is defined as dividing a text in syntactically correlated parts of words [11]. Chunking consists of two processes - first identifying proper chunks from a sequence of tokens (such as words), and second classifying these chunks into some grammatical classes. Major advantages of using text chunking over full parsing techniques are that partial parsing such as text chunking is much faster, more robust, yet sufficient for IE.

Support Vector Machine (SVMs) based text chunking was reported to produce the highest accuracy in a text chunking task [11]. The SVMs-based approach such as other inductive-learning approaches takes as input a set of training examples (given as

binary valued feature vectors) and finds a classification function that maps them to a class.

In general, SVM models can be characterized as follows: First, SVMs are known to robustly handle large feature sets and to develop models that maximize their generalizability. This makes them an ideal model for IE. Generalizability in SVMs is based on statistical learning theory and the observation that is useful to misclassify some of the training data so that the margin between other training points is maximized [5]. This is particularly useful for real world data sets that often contain inseparable data points. Although training is generally slow, the resulting model is usually small and runs quickly as only the patterns that help define the function that separates positive from negative examples. In addition, SVMs are binary classifiers and so we need to combine SVM models to obtain a multiclass classifier.



N denote noun, P denotes preposition, T denotes target, V denote verb

Fig. 3. Noun phrase based Mixture Hidden Markov Models

Due to the nature of the SVM as a binary classifier it is necessary in a multi-class task to consider the strategy for combining several classifiers. In this paper, we use Tiny SVM [5] in that Tiny SVM performs well in handling a multi-class task.

3.2 Relation Extraction by MiHMM Component

Figure 3 is a schematic representation of how our MiHMM works. Our phrase group includes 14 phrase types. Our models are constructed with the assumption that the model is fully connected, which means that the model emits a segment of any type at any given position within the sentence. Bold boxes in Figure 3 indicate the target noun group that contains either proteins or a protein-protein pair. Each box represents phrase group and circles inside the box show the POS tags assigned to words in order that appears in the sentence.

In a generic Hidden Markov Model, it is typical to define a number of states and a number of transitions between those states. The more complex the HMM, the better it can represent a document, but also the more data that is needed to dependably train the model and avoid errors due to noise in the data. Consequently there is an apparent tradeoff between representational efficacy and training efficiency, and this tradeoff varies from domain to domain.

Therefore, rather than deciding on just one model, it is often easier to use a mixture of models and decide later on how much to weight each model. This approach is quite effective because it allows one to model a document at varying degrees of granularity by effectively using a hierarchical model. At the same time it also retains the advantages of each model. That is, if the data is sparse, the simpler model will likely perform better and thus be weighted more during the extraction phase. And if there is an abundance of data, the more complex model can be robustly trained and be weighted more heavily during the extraction phase. For MiHMM, a basic set of three mixture models was used and are shown in Figure 4. A similar mixture model was proposed by [6]. Comparing our approach to Freitag and McCallum's, their model utilizes fairly complex prefix and suffix structures. Despite the simplicity of these models however, the primary benefit of these models that they can be trained on very sparse data.

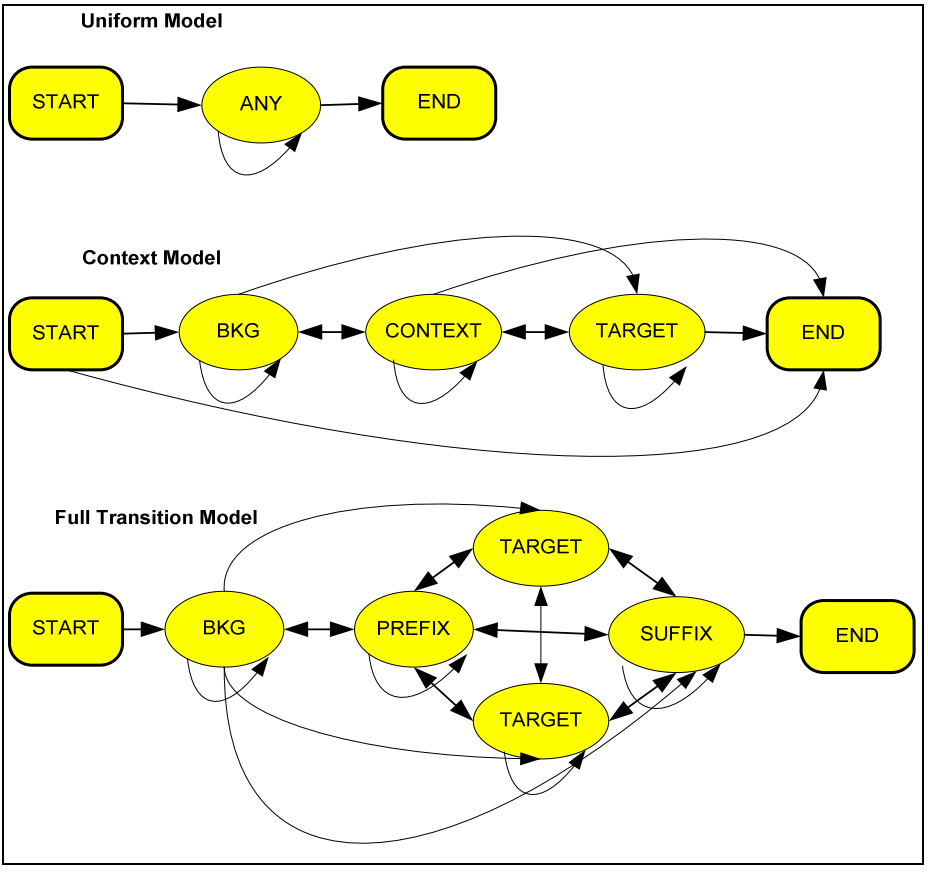
The model is trained with maximum likelihood parameter estimation. From the sentence training set we can easily obtain the information concerning the frequency that a given state or observation occurred and the frequency with which a state transition or observation emission was made.

The parameters of the model are the transition probabilities $P(q \to q')$ that one state follows another and the emission probabilities $P(q \uparrow q')$ that a state emits a particular output symbol. The probability of a string x being emitted by an HMM is computed as a sum over all possible paths by:

$$P(x \mid M) = \sum_{q_1,\ldots q_l \in Q^l} \prod_{k=1}^{l+1} P(q_{k-1} \to q_k) P(q_k \uparrow x_k) \tag{1}$$

where q_0 and q_{l+1} are restricted to be q_I and q_F respectively, and is an end-of-string token.

The forward algorithm can be used to calculate this probability [17]. The observable output of the system is the sequence of symbols that the states emit, but



BKG denotes Background

Fig. 4. Graphic representation of MiHMM

the underlying start sequence itself is hidden. One common goal of learning problems that use HMMs is to recover the state sequence $V(x \mid M)$ that has the highest probability of having produced an observation sequence:

$$V(x \mid M) = \arg\max_{q_1 \ldots q_l \in Q^l} \prod_{k=1}^{l+1} P(q_{k-1} \rightarrow q_k) P(q_k \uparrow x_k) \quad (2)$$

Determining this state sequence is efficiently performed by dynamic programming with the Viterbi algorithm [21].

4 Evaluation

To evaluate KXtractor, we compare it with three other well-known IE methods: 1) the dictionary-based extraction, 2) RAPIER, a rule-based machine learning extraction,

and 3) single POS HMM. Performance of these IE systems is measured by precision, recall, and the F-measure. The data used for experiments are retrieved from MEDLINE.

4.1 Data Collection

The IE task conducted in this paper is a multiple slot extraction task. The goal of our IE task is to extract instances of *n*-ary relations; that is, protein-protein interactions. A MEDLINE record may contain multiple proteins but this relation holds only among certain pairs of these proteins.

The protein-protein interaction data sets are composed of abstracts gathered from the MEDLINE database [13]. MEDLINE contains bibliographic information and abstracts from more than 4000 biomedical journals. From this huge text corpus, we combined and utilized MEDLINE data sets provided by Skounakis et al. [19] and Bunescu et al. [2]. The data sets consist of 1700 MEDLINE records. These data sets characterize physical interactions between pairs of proteins. In terms of sentences, the data sets consist of 6417 positive and 46123 negative sentences. It contains 10123 instances of 913 protein-protein pairs. To label the sentences in these abstracts, we matched the target tuples to the words in the sentence. A sentence that contained words that matched a tuple was taken to be a positive instance. Other setences were considered to be negative instances.

4.2 Dictionary-Based Extraction

We developed a Dictionary-based extraction system proposed by Blaschke et al. [1]. The following six steps were taken to extract protein-protein interactions: 1) the protein names are collected from the Database of Interacting Proteins (DIP) and Protein-Protein Interaction Database (PPID) databases. The synonyms of the target proteins are manually provided. 2) The 14 verbs, indicating actions related to protein interaction, are used. 3) Abstracts are provided from MEDLINE. 4) The passages containing target proteins and actions are identified. 5) The original text is parsed into fragments preceding grammatical separators. 6) The final step is to build protein-protein pairs.

4.3 RAPIER

To evaluate the performance of KXtractor, we compare KXtractor with RAPIER. RAPIER [3] is a well-known IE system that was developed with a bottom-up inductive learning technique for learning information extraction rules. In order to use the slot-filling IE systems like RAPIER for extracting relations, we adapt the Role-filler approach proposed by Bunescu et al. [2].

The Role-filler approach allows for extracting the two related entities into different role-specific slots. For protein interactions, Bunescu and his colleagues [2] name the roles interactor and interactee. As indicated by the role names, protein-protein interactions are defined with the assumption that proteins appear in the same sentence. Bunescu et al. [2] extracted the related pairs using the following criteria: 1) the interactors and interactees appear in the same sentence, 2) each interactor is associated with the next occurring interactee in the segment, and 3) If the number of the interactors

and the interactees are unequal, use the last interactor (interactee) for building the remaining pairs.

4.4 Single POS HMM

In order to verify that our MiHMM models are superior to a simple HMM, we develop a simple HMM, based on single terms and a single model that incorporate less grammatical information. We implemented single-level HMMs whose states emit words, but are typed with part-of-speech (POS) tags so that a give state can emit words with only a single POS. The Viterbi algorithm extracts information from documents modeled by an HMM. With the fix structure, the objective of learning is to give high probabilities to training documents. The result of learning is estimated probabilities for vocabularies and transitions.

4.5 Sample Output

The sample outcome of running KXtractor is illustrated in Table 1. Doc ID indicates the PubMed record ID that contains an abstract that the target protein pairs are extracted from.

Table 1. Sample results of running KXtractor

Doc ID: PUBMED8681382 Sentence ID: 1 Target Protein 1: yta10p Target Protein 2: yta12p
Doc ID: PUBMED8182122 Sentence ID: 5 Target Protein 1: spo7p Target Protein 2: nem1p

Sentence ID is the ID assigned to the sentence by KXtractor in the PubMed record. Target protein 1 and target protein 2 indicates the protein pairs that KXtractor extracts for the task of protein-protein interaction.

5 Experiments

We conducted experiments to evaluate the performance of KXtractor on the task of protein-protein interaction extraction. In experiments the machine learning systems

were trained using the abstracts with proteins and their interactions, processed by the text chunking technique. With these set of data, the IE systems extract interactions among these proteins. This gives us a measure of how the protein interaction extraction systems alone perform.

Performance is evaluated using ten-fold cross validation and measuring recall and precision. As the task of interest is only to extract interacting protein-pairs, in our evaluation we do not consider matching the exact position or every occurrence of interacting protein-pairs within the abstract.

To evaluate our IE systems, we construct a precision-recall graph. Recall denotes the ratio of the number of slots the system found correctly to the number of slots in the answer key, and precision is the ratio of the number of correctly filled slots to the total number of slots the system filled.

Table 2. Comparison of extraction system performance

Extraction System	***Precision***	***Recall***	***F-Measure***
Dictionary-based extraction	62.31%	32.81%	36.10%
RAPIER	60.17%	34.12%	35.49%
Single POS HMM	67.40%	47.23%	43.80%
KXtractor	70.23%	51.21%	52.38%

Our experiments show that RAPIER produces relatively high precision but low recall. The similar results are observed in the dictionary-based extraction method which gives also high precision but low recall. Single POS HMM produces the second best results, although recall is relatively lower than precision. Among these three systems, KXtractor outperforms RAPIER, Dictionary, and single POS HMM in terms of precision, recall, and the F-measure. As shown in Table 2, the F-Measure of KXtractor is 52.38% whereas RAPER is 35.49%, dictionary is 36.10%, and single POS HMM is 43.80%.

Figure 5 shows the precision-recall graphs of KXtractor, RAPIER, Dictionary, and single POS HMM-based extraction for the protein-protein interaction data set. The curve for KXtractor is superior to the curves for RAPIER, Dictionary, and single POS HMM.

We repeated the same experimental tests over the five different datasets. Figure 6 shows the results of the four extraction methods, KXtractor, Single POS HMM, RAPIER, and Dictionary in F-Measure. KXtractor outperforms the other three algorithms. Accuracy of KXtractor ranges between 51.23% and 59.36% in F-Measure. Single POS HMM ranges between 43.8% and 45.93% in F-Measure. RAPIER ranges between 35.49% and 38.84% in F-Measure. Dictionary ranges between 36.1% and 39.+95% in F-Measure.

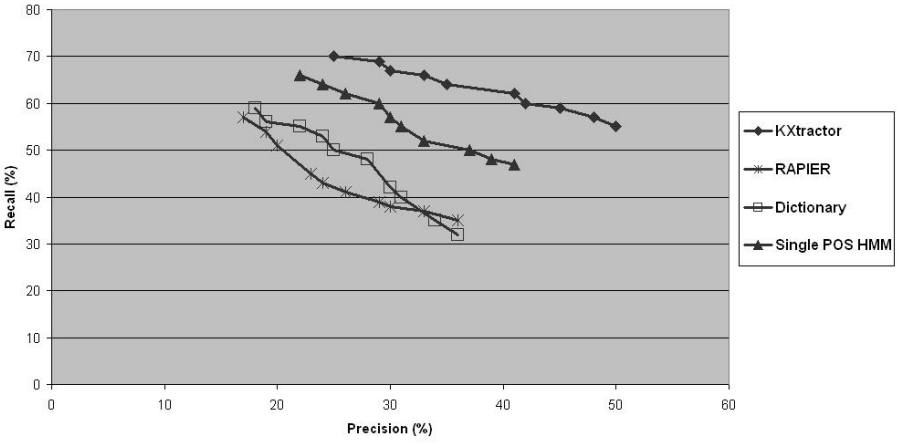


Fig. 5. Precision-recall graph for extracting protein-protein pairs

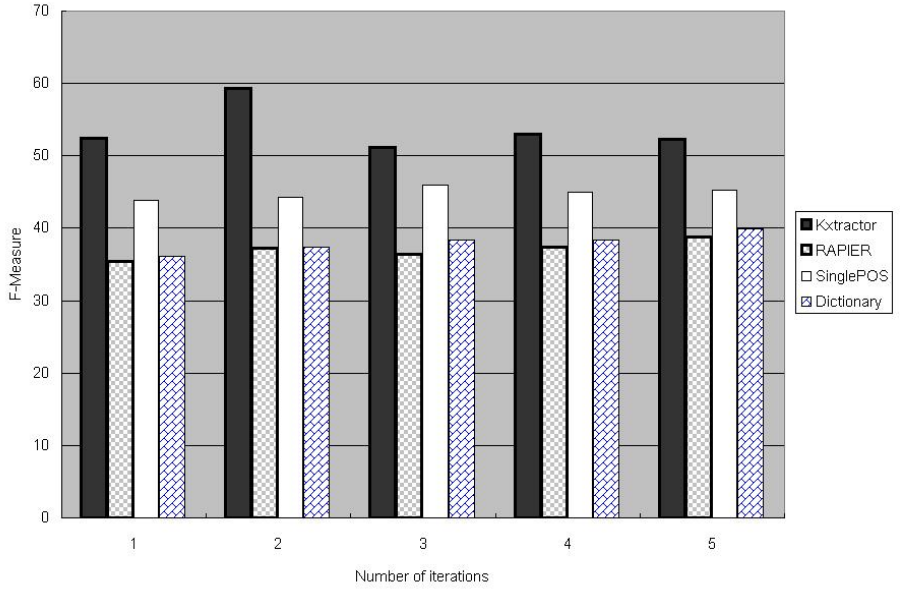


Fig. 6. Performance Comparison of Four Extraction Algorithmsover Five Different Data sets

6 Conclusion

In this paper, we proposed a novel and high quality information extraction system, called KXtractor, a noun phrase-based Mixture Hidden Markov Models (MiHMM) system.

KXtractor is differentiated from other approaches in that (a) It overcomes the problem of the single POS HMMs with modeling the rich representation of text where features overlap among state units such as word, line, sentence, and paragraph. By incorporating sentence structures into the learned models, KXtractor provides better extraction accuracy than the single POS HMMs. (b) It resolves the issues with the single POS HMMs for IE that operate only on the semi-structured such as HTML documents and other text sources in which language grammar does not play a pivotal role.

KXtractor consists of two major components: 1) text chunking and 2) Mixture Hidden Markov Models (MiHMM) component. The text chunking component groups the sentence with Support Vector Model (SVM) technique. MiHMM takes a set of sentences processed by the text chunking technique. MiHMM then learns a generative probabilistic model of the underlying state transition structure of the sentence from a set of tagged training data. Given a trained probabilistic mixture model of the data, the system applies this model to new (unseen) input documents to predict which portions of these sentences are likely targets according to the training data template.

We compared KXtractor with three well-known IE techniques: 1) RAPIER, a rule-based machine learning system, 2) Dictionary-based extraction system which was proposed by Blaschke at al. [1], and 3) single POS HMM. Our experiments showed that KXtractor outperforms other IE techniques such as RAPIER, dictionary-based, and single POS HMM in extracting protein-protein interactions in terms of F-measure. The F-Measure of KXtractor is 52.38% whereas RAPER is 35.49%, dictionary is 36.10%, and single POS HMM is 43.80%. In addition, both precision and recall of KXtractor are higher than those of RAPIER, Dictionary, and single POS HMM.

In follow-up papers, we will apply KXtractor to other types of relation extractions such as subcellular-localization relation extraction. We also plan to compare KXtractor with other IE systems such as MaxEnt and SVM.

References

1. Blaschke, C., Andrade, M.A., Ouzounis, C., and Valencia, A. (1999). Automatic Extraction of Biological Information from Scientific Text: Protein-Protein Interactions, *In Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, Heidelberg, Germany, 60-67.
2. Bunescu, R., Ge, R., Kate, R.J., Marcotte, E.M., Mooney, R.J., Ramani, A.K., and Wong, Y.W. (2004). Comparative Experiments on Learning Information Extractors for Proteins and their Interactions. To appear in *Journal Artificial Intelligence in Medicine (Special Issue on Summarization and Information Extraction from Medical Documents).*
3. Califf, M.E. and Mooney, R.J. (1999). Relational Learning of Pattern-Match Rules for Information Extraction. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, FL, 328-334.
4. Collier, N., Nobata,C., and Tsujii, J. (2000). Extracting the Names of Genes and Gene Products with a Hidden Markov Model. *Proceedings of the 18th International Conference on Computational Linguistics (COLING2000)*, Saarbrucken, Germany, 201-207.
5. Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, 20(3): 273-297.
6. Freitag, D. and McCallum, A. (1999). Information extraction with HMMs and shrinkage. AAAI-*99 Workshop on Machine Learning for Information Extraction*, Orlando, FL, 31-36.

7. Friedman, C., Kra, P., Yu, H.,Krauthammer, M., and Rzhetsky, A. (2001). GENIES: a natural-language processing system for the extraction of molecular pathways from journal articles. *Bioinformatics*, 17 Suppl 1, S74-82.
8. Fukuda, K., Tamura, A., Tsunoda, T., and Takagi, T. (1998). Toward information extraction: identifying protein names from biological papers. *Pacific Symposium Biocomputing*. 707-18.
9. Jenssen, T.K., Laegreid, A., Komorowski, J., and Hovig, E. (2001). A literature network of human genes for high-throughput analysis of gene expression. *Nature Genetics*, 28(1): 21-8.
10. Krauthammer, M., Kra, P., Iossifov,I., Gomez, S. M., and Hripcsak, G. (2002). Of truth and pathways: Chasing bits of information through myriads of articles. *Bioinformatics*, 18 Suppl 1, S249-S257.
11. Kudo, T. and Matsumoto, Y. (2000). Use of Support Vector Learning for Chunk Identification. In Proceedings of CoNLL- 2000 and LLL-2000, Saarbruncken, Germany, 142-144.
12. Leek, T. R. (1997). *Information extraction using Hidden Markov Models*. MSc Thesis, Department of Computer Science, University of California, San Diego.
13. National Library of Medicine (2003). *The MEDLINE database*, http://www.mcbi.nlm. nih.gov/PubMed/.
14. Proux, D., Rechenmann, F., and Julliard, L. (2000). A pragmatic information extraction strategy for gathering data on genetic interactions. *Proceedings of International Conference on Intelligent System for Molecular Biology*, La Jolla, CA, 8:279-85.
15. Pustejovsky, J., Castano, J., Zhang, J., Kotecki, M., and Cochran, B. (2002). Robust relational parsing over biomedical literature: extracting inhibit relations. *Pacific Symposium on Biocomputing*, 362-73.
16. Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257-286.
17. Shatkay H. and Feldman R. (2003). Mining the biomedical literature in the genomic era: An overview. *Journal of Computational Biology*, 10 (6): 821-855.
18. Skounakis, M., Craven, M., and Ray, S. (2003). Hierarchical Hidden Markov Models for Information Extraction. *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August, 427-433.
19. Song, M, Song, I-Y., and Hu, X. (2003). KPSpotter: A Flexible Information Gain-based Keyphrase Extraction System, *Fifth International Workshop on Web Information and Data Management (WIDM'03)*, New Orleans, LA, 50-53.
20. Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Processing*, 13:260-269.

Phylogenetic Networks: Properties and Relationship to Trees and Clusters

Luay Nakhleh[1] and Li-San Wang[2]

[1] Department of Computer Science, Rice University, Houston, TX 77005, USA
nakhleh@cs.rice.edu
[2] Department of Biology, University of Pennsylvania, Philadelphia, PA 19104, USA
lswang@mail.med.upenn.edu

Abstract. Phylogenetic networks model evolutionary histories in the presence of non-treelike events such as hybrid speciation and horizontal gene transfer. In spite of their widely acknowledged importance, very little is known about phylogenetic networks, which have so far been studied mostly for specific datasets.

Even when the evolutionary history of a set of species is non-treelike, individual genes in these species usually evolve in a treelike fashion. An important question, then, is whether a gene tree is "contained" inside a species network. This information is used to detect the presence of events such as horizontal gene transfer and hybrid speciation. Another question of interest for biologists is whether a group of taxa forms a clade based on a given phylogeny. This can be efficiently answered when the phylogeny is a tree simply by inspecting the edges of the tree, whereas no efficient solution currently exists for the problem when the phylogeny is a network. In this paper, we give polynomial-time algorithms for answering the above two questions.

1 Introduction

Phylogenies are the main tool for representing the relationships among biological entities. Their pervasiveness has led biologists, mathematicians, and computer scientists to design a variety of methods for their reconstruction. Furthermore, extensive studies have been focused on the performance of these methods under different models and settings, as well as on the combinatorial and biological properties of trees (e.g., [7, 2]). However, almost all such methods construct trees, and almost all studies have been aimed at trees. Yet, biologists have long recognized that trees oversimplify our view of evolution, since they cannot take into account such events as hybridization, lateral gene transfer, and recombination. These non-tree events give rise to edges that connect nodes on different branches of a tree, giving rise to a directed acyclic graph structure that is usually called a *phylogenetic network*.

A gene tree is a model of how a gene evolves through duplication, loss, and nucleotide substitution. Gene trees can differ from one another as well as from the species phylogeny. Such differences arise during the evolutionary process due to events such as duplication and loss, whereby each genome may end up with multiple copies of a given gene—but not necessarily the same copies that survive in another genome. Unless the genome is very well sampled, only a subset (sometimes only one copy, in fact) of the

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 82–99, 2005.

gene is used in phylogenetic analyses. As a result, the phylogeny for the gene may not agree with the species phylogeny, nor with the phylogeny for another gene. Because the gene copy has a single ancestral copy, barring recombination, the resulting history is a branching tree. Point mutations can cause some of the copies to be imperfect representations of the original, but this process does not compromise the existence of the (gene) tree. Events such as recombination, hybrid speciation, and lateral gene transfer break up the genomic history into many small pieces, each of which has a strictly treelike pattern of descent [4]. Thus, within a species phylogeny, many tangled gene trees can be found, one for each nonrecombined locus in the genome. Incongruence among gene trees is a powerful tool for detecting recombination, hybrid speciation, and other non-treelike evolutionary events (e.g., see [6]). While testing for incongruence between two (gene) trees can be done in a straightforward manner, it is not as simple for testing the incongruence between a tree and a network, since the number of trees "inside" a network grows exponentially with the number of non-treelike events. In this paper, we give the first polynomial-time algorithm for solving this problem.

A phylogeny can be viewed as a collection of clusters of taxa (each defined as the set of leaves in a subtree). Various approaches for reconstructing phylogenies (trees and networks) have been proposed based on this view (see, e.g., [1, 3]). An interesting biological question, then, is whether a group of taxa forms a cluster in a given phylogeny. This question can be answered in a straightforward manner when the phylogeny is a tree, since each edge in a tree defines a unique cluster. However, the number of clusters in a phylogenetic network grows exponentially with the number of non-treelike events, and hence an efficient algorithm for solving the problem is not straightforward. In this paper, we present the first polynomial-time algorithm for solving this problem.

The rest of the paper is organized as follows. In Section 2 we give a background on trees, clades and clusters. In Section 3 we briefly describe evolutionary events that necessitate phylogenetic networks, and describe the graph-theoretic model of phylogenetic networks that we use in the paper, along with combinatorial properties that follow from the model. In Section 4 we introduce the concepts of network decomposition and dependency graphs. computing these structures forms the core of our algorithms. In Section 5, we define reduced inheritance profiles and present the main lemma on which the our algorithms are based. In Section 6 we describe our polynomial-time algorithms for solving the aforementioned decision problems. We conclude in Section 7 with a summary of our main results and directions for future research.

2 Background: Phylogenetic Trees

2.1 Notation

In this paper, and unless stated otherwise, all graphs are directed. Given a graph G, $E(G)$ denotes the set of (directed) edges of G and $V(G)$ denotes the set of nodes of G. We write (u, v) to denote a directed edge from node u to node v. If $e = (u, v)$ is an edge from u to v, we call u the *tail* and v the *head* of the edge and say that u is a *parent* of v. The *indegree* of a node v, denoted $indeg(v)$, is the number of edges whose head is v, while the *outdegree* of v, denoted $outdeg(v)$, is the number of edges whose tail is v. The degree of a node v is the sum of its indegree and outdegree. In an undirected graph,

the degree of a node v is the number of edges incident with v. A node u is *redundant* if $indeg(u) = outdeg(u) = 1$. A directed path of length k from u to v in G is a sequence $u_0 u_1 \cdots u_k$ of nodes with $u = u_0$, $v = u_k$, and $\forall i,\ 1 \leq i \leq k$, $(u_{i-1}, u_i) \in E(G)$. Node v is *reachable* from u in G, denoted $u \rightsquigarrow v$, if there is a directed path in G from u to v.

2.2 Phylogenetic Trees, Bipartitions and Clusters

A *phylogenetic tree* is a leaf-labeled tree that models the evolution of a set of taxa (species, genes, languages, placed at the leaves) from their most recent common ancestor (placed at the root). The internal nodes of the tree correspond to the speciation events.

Mathematically, A *rooted* phylogenetic tree is a rooted tree without redundant nodes and whose leaves are labelled distinctively. An unrooted phylogenetic tree is a rooted phylogenetic tree with the root suppressed. Every edge e in an unrooted leaf-labeled tree T defines a bipartition (or, split) $\pi(e)$ on the leaves (induced by the deletion of e), so that we can define the set $\Pi(T) = \{\pi(e) : e \in E(T)\}$. Every edge e in a rooted leaf-labeled tree T defines a cluster $c(e)$ of leaves (those leaves that are reachable from the root through e), so that we can define the set $C(T) = \{c(e) : e \in E(T)\}$. A clade of a rooted tree T is the entire subtree rooted at a node of T; the set of all leaves in a clade correspond to a cluster of T.

There is a many-to-one relationship between rooted and unrooted phylogenetic trees: there are many ways to root an unrooted phylogenetic tree. Based on this we also see an association between clusters of a rooted tree T and bipartitions of the unrooted version of T: each cluster of a rooted tree T equals one of the two sets in the bipartition induced by an edge e in the unrooted version of T.

3 Phylogenetic Networks

3.1 Non-tree Evolutionary Events

We now describe two types of evolutionary events that give rise to network (as opposed to tree) topologies: hybridization and lateral gene transfer. In hybridization, two lineages recombine to create a new species, as symbolized in Figure 1(a). We can distinguish between *diploid hybridization*, in which the new species inherits one of the two homologs for each chromosome from each of its two parents—so that the new species has the same number of chromosomes as its parents, and *polyploid hybridization*, in which the new species inherits the two homologs of each chromosome from both parents—so that the new species has the sum of the numbers of chromosomes of its parents. Prior to hybridization, each site on each homolog has evolved in a tree-like fashion, although, due to meiotic recombination, different strings of sites may have different histories. Thus, each site in the homologs of the parents of the hybrid evolved in a tree-like fashion on one of the trees contained inside (or induced by) the network representing the hybridization event, as illustrated in Figures 1(b) and 1(c). In lateral gene transfer, genetic material is transferred from one lineage to another without resulting in the production of a new lineage, as symbolized in Figure 1(d). In an evolutionary scenario involving lateral transfer, certain sites are inherited through lateral transfer from

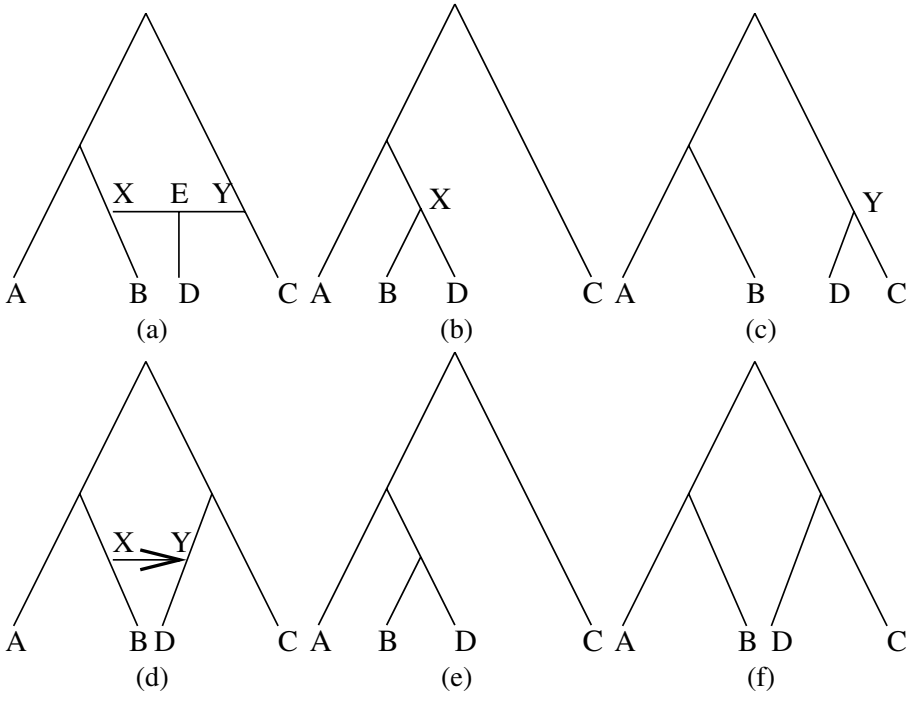


Fig. 1. Hybrid speciation: the network in (a) and its two induced trees in (b) and (c). Horizontal transfer: the network in (d) and its two induced trees in (e) and (f).

another species, as in Figure 1(e), while all others are inherited from the parent, as in Figure 1(f).

When the evolutionary history of a set of taxa involves processes such as hybridization or lateral gene transfer, trees can no longer represent the evolutionary relationship; instead, we turn to rooted directed acyclic graphs (rooted DAGs).

3.2 Phylogenetic Networks: Model and Properties

In this paper, we adopt the general model of (reduced) phylogenetic networks, as described in [5]).

Definition 1. *A* **phylogenetic network** *is a connected directed acyclic graph* $N = (V, E)$*, where* V *can be partitioned into* $\{r\} \cup Tr(N) \cup Nt(N) \cup L(N)$*, where:*

1. *Node* r *is the root; it has indegree* 0.
2. *Set* $Tr(N)$ *is the set of tree nodes; each node* u *in* $Tr(N)$ *has indegree* 1 *and outdegree* > 1.
3. *Set* $Nt(N)$ *is the set of network nodes; each node* v *in* $Nt(N)$ *has indegree* 2 *and outdegree* 1.

4. *Set $L(N)$ is the set of leaf nodes (taxa); each node x in $L(N)$ has indegree 1 and outdegree 0. Each node x in $L(N)$ is labeled uniquely by an integer i, where $1 \leq i \leq |L(N)|$.*

Figures 1(a) and 1(d) give two examples of phylogenetic networks. Given a network N, we classify its edges as *tree edges* and *network edges*. An edge $e = (u, v)$ is a tree edge if v is a tree node or a leaf; otherwise, it is a network edge. Biologically, the tree nodes correspond to regular speciation events in the evolutionary history, whereas network nodes correspond to reticulation events (e.g., hybridization, lateral gene transfer, recombination, etc.).

We say that network N is *binary* if the root and all tree nodes of N have outdegree 2. In this paper, and unless noted otherwise, all networks are binary. Further, we assume that if u is a tree node and (u, v) and (u, w) are the two edges incident from u, then at least one of the two nodes v and w is a tree node.

A *forced contraction* is an operation on a graph in which we delete a redundant node and replace the two edges incident to it by a single edge. An *augmentation* is an operation on a graph in which an edge (u, v) is replaced by two edges (u, x) and (x, v), where x is a new node. A DAG N is a *pseudo-network* if a network N' can be obtained by applying a sequence of forced contraction operations to N (alternately, if N can be obtained by applying a sequence of augmentation operations to a network N'). We generalize the *clade* concept to networks as follows. Given a network N, we say that the DAG N', rooted at node x, is a *network clade* of N, if there exists an edge $e = (u, x)$ in N whose removal disconnects N, thus creating two components, one of which is N' (rooted at x). If network clade N' does not contain network nodes, i.e., N' is a tree, we refer to N' simply as a clade. Given a network N, and a clade N', we say that N' is *maximal* if N does not contain any clade N'' such that $N' \subset N''$.

A phylogenetic network $N = (V, E)$ defines a partial order on the set V of nodes, and based on this partial order, we assign times to the nodes of N; $t(u)$ denotes the time associated with node u. If there is a directed path p from node u to node v, such that p contains at least one tree edge, then $t(u) < t(v)$. If $e = (u, v)$ is a network edge, then $t(u) = t(v)$ (since reticulation events occur instantaneously). Further, if there is a directed path from node u to node v, $u \neq v$, we say that u is *above* v and that v is *below* u, both denoted by $u > v$. We say that node u in N is a *lowest network node* if (1) u is a network node, and (2) for any network node v, $v \neq u$, we have $u \not\rightsquigarrow v$.

Lemma 1. *Let N be a network, u be a lowest network node, and $e = (u, v)$ be the edge incident from u. Then, the subgraph $N' \subset N$ rooted at v is a maximal clade.*

Proof. By definition of lowest network node, all nodes below u are either tree nodes or leaves; hence, N' is a clade. Assume N'' is also a clade, and that $N' \subset N''$. Then, N'' contains node u, which is a network node – a contradiction. Therefore, N' is a maximal clade.

Given a network N and two nodes u and v in N, we say that u and v cannot co-exist in time if there is a directed path $p = \langle u_0, u_1, \ldots, u_k \rangle$ in N, where $u_0 = u$ and $u_k = v$, and p satisfies three properties: (1) p contains at least one tree edge, (2) for any tree edge e on p, we have $e = (u_i, u_{i+1})$ (may not be (u_{i+1}, u_i)), $0 \leq i \leq k - 1$, and (3) the orientation of a network edge on p is irrelevant.

Since events such as hybridization and lateral gene transfer occur between two lineages (nodes in the network) that co-exist in time, a phylogenetic network N must satisfy the *synchronization property*, which states that if two nodes x and y cannot co-exist in time, then there do not exist two edges $e = (x, v)$ and $e' = (y, v)$ in N. If a network N violates the synchronization property (which may happen due to missing taxa in the phylogenetic analysis), then N can be augmented to remedy this violation, as we show in the following theorem.

Theorem 1. *For any phylogenetic network N, there exists an augmentation of N into a pseudo-network N' that satisfies the synchronization property.*

Proof. If N satisfies the synchronization property, then $N' = N$. Assume N does not satisfy the synchronization property, and let $e_1 = (x_1, v)$ and $e_2 = (x_2, v)$ be two network edges such that $x_1 \rightsquigarrow x_2$. Let N' be the network obtained from N by replacing edge e_1 by two new edges $e'_1 = (x_1, y)$ and $e''_1 = (y, v)$, where y is a new node. Now, the network node v has the two parents y and x_2. It is clear that $x_2 \not\rightsquigarrow y$, since the only way to reach y is through x_1, and $x_2 \not\rightsquigarrow x_1$ (otherwise, N would be cyclic). It is also clear that $y \not\rightsquigarrow x_2$, since if y reaches x_2, it has to be via a path that passes through v, and since x_2 reaches v, N would be cyclic. We apply the same process to every pair of edges that violates the synchronization property.

We write $N|_{L'}$, where $L' \subset L(N)$, to denote the subgraph N' obtained from N by removing all leaves not in L', and then applying forced contraction operations and removal of nodes of outdegree 0 (other than the leaves in L'). We now describe some properties of phylogenetic networks.

Proposition 1. *Let $N = (V, E)$ be a phylogenetic network.*

1. *$outdeg(r) + \sum_{s \in Tr(N)}(outdeg(s) - 1) = |Nt(N)| + |L(N)|$.*
2. *For every node $v \in V$, $r \rightsquigarrow v$.*
3. *For every node $v \in V$, there exists at least one leaf l below v.*
4. *(Taxon sampling) $N|_{L'}$ is a phylogenetic network, for any $L' \subset L(N)$.*

Proof.

1. By the observation $\sum_{v \in V} outdeg(v) = \sum_{v \in V} indeg(v)$.
2. Let V' be the set of all nodes that cannot be reached from r. Let x be a maximal element (in terms of the partial order induced by N on V) in V'; then $indeg(x) = 0$ (otherwise N would be cyclic). However, the only node with indegree 0 is r – a contradiction.
3. Let $R(v) = \{u \in V : v > u\}$. If $R(v) = \emptyset$, then $outdeg(v) = 0$, i.e., v is a leaf. If $R(v) \neq \emptyset$, and since N is acyclic (and finite), then there exists at least one node x in $R(v)$ with outdegree 0. It follows from Definition 1 that x is a leaf.
4. Straightforward.

In this paper, we focus on binary networks, but the results extend to general networks in a straightforward manner.

3.3 Networks and Trees

There is a fundamental connection between (species) networks and (gene) trees. A gene tree is a model of how a gene evolves through duplication, loss, and nucleotide substitution. Gene trees can differ from one another as well as from the species phylogeny. Such differences arise during the evolutionary process due to events such as duplication and loss, whereby each genome may end up with multiple copies of a given gene—but not necessarily the same copies that survive in another genome. Unless the genome is very well sampled, only a subset (sometimes only one copy, in fact) of the gene is used in phylogenetic analyses. As a result, the phylogeny for the gene may not agree with the species phylogeny, nor with the phylogeny for another gene. Because the gene copy has a single ancestral copy, barring recombination, the resulting history is a branching tree. Point mutations can cause some of the copies to be imperfect representations of the original, but this process does not compromise the existence of the (gene) tree. Events such as recombination, hybridization, and lateral gene transfer break up the genomic history into many small pieces, each of which has a strictly treelike pattern of descent [4]. Thus, within a species phylogeny, many tangled gene trees can be found, one for each nonrecombined locus in the genome. Yet, in the presence of these processes, the evolutionary history of the species fails to be modeled as a tree; in this case, networks are used to model the species phylogeny. We say that a (species) network "induces" (or, contains) a (gene) tree or alternately, a (gene) tree is *induced by* (or, contained inside) a (species) network. We formalize this concept as follows.

Let N be a network with p network nodes $h_1, h_2, \ldots, h_p$. Further, assume that the two edges incident into h_i are e_{i_1} and e_{i_2}. An *inheritance profile*, IP, for N is a set of size p and which contains exactly one of the two edges e_{i_1} and e_{i_2} for each network nodes h_i. A rooted tree T is *induced* by (or, contained in) a network N if there exists an inheritance profile IP such that T can be obtained from N as follows: for network node h_i, if $e_{i_1} \in IP$, remove edge e_{i_2}; otherwise, remove edge e_{i_1}. (and then apply forced contraction operations to the resultant graph). Biologically, the evolutionary history of a gene within the species network corresponds to a tree T induced by N. Associated with this tree is an inheritance profile IP that decides how to obtain T from N; in this case, we say that IP is a *valid* inheritance profile that induces T. A network N induces (or, contains) a cluster C, $C \subseteq L(N)$, if there exists a tree T such that N induces T and C is a cluster of T.

Proposition 2. *Let N be a nonempty network. Then N induces at least one phylogenetic tree.*

Proof. We show the proposition by induction on the number of leaves in N. The base case (one leaf) is trivial. Assume the hypothesis is true for $|L(N)| = n$, and consider the case where $|L(N)| = n + 1$. Let N_n be the DAG obtained by restricting N to the first n leaves. By the induction hypothesis, there exists a tree T_n that is induced by N_n. By Proposition 1, there exists a path P_{n+1} connecting the root and leaf $n + 1$, and there exists a node v that is the lowest node in both P_{n+1} and the embedding of T_n in N_{n+1}. T is obtained by joining the edges and nodes below v in P_{n+1} and T_n. Since T is connected by construction, if T is not a tree, then there exists a (not necessarily oriented) cycle in T. This contradicts the choice of v as the lowest node in P_{n+1}.

As mentioned before, deciding whether a cluster or a tree are induced by a given network plays a significant role in solving major problems such as network reconstruction, gene tree and species network relationships, exploring the network space in hill-climbing heuristics for solving hard optimization network reconstruction problems, measuring distances and error rates between networks in simulation studies, and many other tasks. We now formalize the two decision problems.

Problem 1. (THE NETWORK-TREE CONTAINMENT PROBLEM)

Input: A phylogenetic network N and a tree T.
Question: Does N contain T?

Problem 2. (THE NETWORK-CLUSTER CONTAINMENT PROBLEM)

Input: A phylogenetic network N and a cluster C.
Question: Does N contain C?

A trivial approach for solving the Network-Cluster Containment Problem is to find "the" lowest common ancestor, x, of C in the network N, and test whether the cluster is contained in the network clade rooted at x. This approach may fail for at least two reasons: (1) x may not be unique in a network, and (2) the network clade rooted at x may contain many of the network nodes of N, in which case the search for a solution would take time that is exponential in the number of network nodes, and hence, probably the network size.

In Section 6 we show that these two problems can be decided in polynomial time. In order to obtain these results, we first introduce the concept of *network decomposition* which forms the basis for our algorithms.

4 Network Decomposition

Before we give the technical details of our algorithms, we describe the network representation we use, which is vital for achieving the running times of the algorithms in the next sections. We assume that a network N is represented using an $n \times n$ adjacency matrix M^N, where n is the number of nodes in the network. We have $M^N[u, v] = 1$ if there is an edge $(u, v) \in E(N)$, and $M^N[u, v] = 0$ otherwise. Using this representation, a forced contraction operation takes $O(1)$ time, and an edge deletion takes $O(1)$ time, as well.

4.1 Preprocessing Networks

An *SH-loop* (speciation-hybridization) is a cycle that contains only network edges, and that consists of two paths p_1 and p_2, such that p_1 and p_2 starting from the same tree node v_0, pass through two sets of network nodes, and end at the same network node v_1. Let $e_1 = (v_0, x)$ and $e_2 = (v_0, y)$ be the two network edges incident from v_0. We break the SH-loop by removing either e_1 or e_2, and applying forced contraction operations to all redundant nodes. We repeat the same process until N is SH-loop-free, i.e., N does not contain any SH-loops.

Preprocessing of a network N can be achieved in polynomial time. To preprocess a network, cycles of network edges in the network need to be detected. A depth-first search achieves this goal. There are at most $\min\{|Tr(N)|, |Nt(N)|\}$ such cycles. Breaking each cycle by removing an edge, followed by a forced contraction operation, takes $O(1)$. Therefore, the overall running time of the preprocessing is $O(|V(N)|(|V(N)| + |E(N)|)) = O(|V(N)|^2) = O(|E(N)|^2)$ (since in binary networks $|V(N)| = \Theta(|E(N)|)$). The following result shows that preprocessing a network N does not change the set of trees induced by N.

Proposition 3. *Let N be a phylogenetic network, and let N' be the network obtained after the preprocessing. Then, $\mathcal{T}(N) = \mathcal{T}(N')$.*

Proof. Since N' is a subgraph of N, we only need to show that each tree T that is induced by N is also induced by N'. As each step in the preprocessing removes one edge from N, it suffices to show this is true if N' and N differ by one edge. Consider an inheritance profile IP that induces T; if each edge in IP is in N', we are done. Otherwise the edge is in an SH-loop of N, pointing from a tree node v_0 to a network node x (x may be suppressed in N'). Let y be the other node immediately below v_0 in the SH-loop, and let v_3 be the lowest network node where the two paths of the SH-loop meet. Let p_1 and p_2 be the two paths in the SH-loop containing edges (v_0, x) and (v_0, y), respectively. Let z and w be the vertices immediately above v_3 in p_1 and p_2, respectively. Notice that since p_1 and p_2 consist of network nodes only (except for v_0), the leaf sets below x, y, and v_3 are identical; call it L. Let L' be the subset of leaves such that the path to root from each leaf in L' passes through (v_0, x) (it also must pass through v_3); such path necessarily passes through (x, y); we only need to consider the case when L' is nonempty. Then IP contains every edge in p_1, and no leaf in L' reaches the root through nodes only in p_2 in T. Hence we can add all edges in p_2 and remove conflicting edges in IP, as they do not lead to leaves from the root. The result is an inheritance profile for N' that also induces T.

4.2 Maximal Clades and Connections

Unless noted otherwise, all networks are SH-loop free. Given a phylogenetic network N, we seek to decompose N into maximal-size clades and disjoint subgraphs of N that connect those clades. To formalize this, we first define some concepts.

Given a node x in network N, we say that a network node y ($y \neq x$) in N is *x-convergent* if any directed path from y to a leaf of N passes through x. Given a maximal clade A of N, and the root a of A, we say that subgraph J of N is the *connection* of A if J is the subgraph obtained by restricting N to all a-convergent nodes and their incident edges.

Lemma 2. *Let A and J be a clade and its connection, respectively, in a network N. Then, when reversing the orientation of its edges, J has a rooted tree topology, where each leaf is a tree node in N and each internal node is a network node in N. Further, the root of J is a lowest network node.*

Proof. Let a be the root of clade A. Assume J has a node v that is a tree node in N. By Proposition 1 and the definitions of J and a, there does not exist a tree node v' that

is reachable from v but not from a; hence, there exists a directed path from v to a and that consists of network nodes only. Moreover, the path is unique. If there exist two such paths from v to a then the two paths form an SH-loop. Consider the union of all such paths to a from speciation nodes in J; since those paths are unique, it follows that, when the orientation of the edges in J is reversed, J forms a rooted tree, and the leaves of J are the set of nodes in J that are tree nodes in N. Other properties follow directly.

4.3 Computing the Decomposition

We now define the concept of *T-decomposition* (tree decomposition) of a network.

Definition 2. *A* T-decomposition *of a network N is an ordered set of pairs $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$, where A_i and J_i are a maximal clade and its connection, respectively, in N_i (N_i is obtained by removing the subgraphs A_{i-1} and J_{i-1} from N_{i-1}, except for the leaves of J_{i-1}, i.e., the tree nodes, and applying forced contraction operations to the resultant graph; for the base case, $N_1 = N$); m is the cardinality of the decomposition.*

Figure 2(b) shows a T-decomposition of the network in Figure 2(a). Before computing a T-decomposition of a network, the network has to be preprocessed as described in Section 4.1.

Definition 2 leads to an algorithm naturally. To compute A_i in N_i, we find a maximal clade, which is rooted at the tree node immediately below a lowest network node (based on Lemma 1). To compute J_i, we use Lemma 2: reverse the orientation of all edges in N_i, do a depth-first search starting from the root of A_i until tree nodes are encountered. J_i is the search tree together with the tree nodes immediately above (and edges connecting them to J_i). This algorithm can be achieved in $O(|Nt(N)||V(N)|)$ time. To find (A_i, J_i) in N_i, we first find a lowest network node v. To find v, we rank the network nodes (a node has a lower rank if it is closer to the root) using topological sort ($O(|V(N)| + |E(N)|)$ running time). We keep a doubly-linked list to allow constant running time update whenever a network node is deleted from the network, so finding a lowest network node can be achieved at no extra cost. The maximal clade A_i is the clade rooted at the node immediately below v. To find the connection J_i, we start from v and do a depth-first search with all edges in N_i reversed, and stop whenever a tree node is encountered. We then remove A_i and J_i from N_i, apply forced contraction to all redundant nodes encountered in the DFS step for finding J_i. Notice that in the two steps for finding A_i and J_i, we visit each edge at most once in component A_i and J_i. These edges are removed from N_i, and the tree nodes in J_i are suppressed in N_i (which takes constant time per node). The overall running time is thus $O(m(|V(N)| + |E(N)|)) = O(|Nt(N)||V(N)|)$.

We now show some properties of the T-decomposition.

Proposition 4. *Let $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$ be a T-decomposition of a network N.*

1. *N_m is a phylogenetic tree.*
2. *Each edge in N belongs to exactly one component in the decomposition.*
3. *Each network node belongs to exactly one connection in the decomposition.*

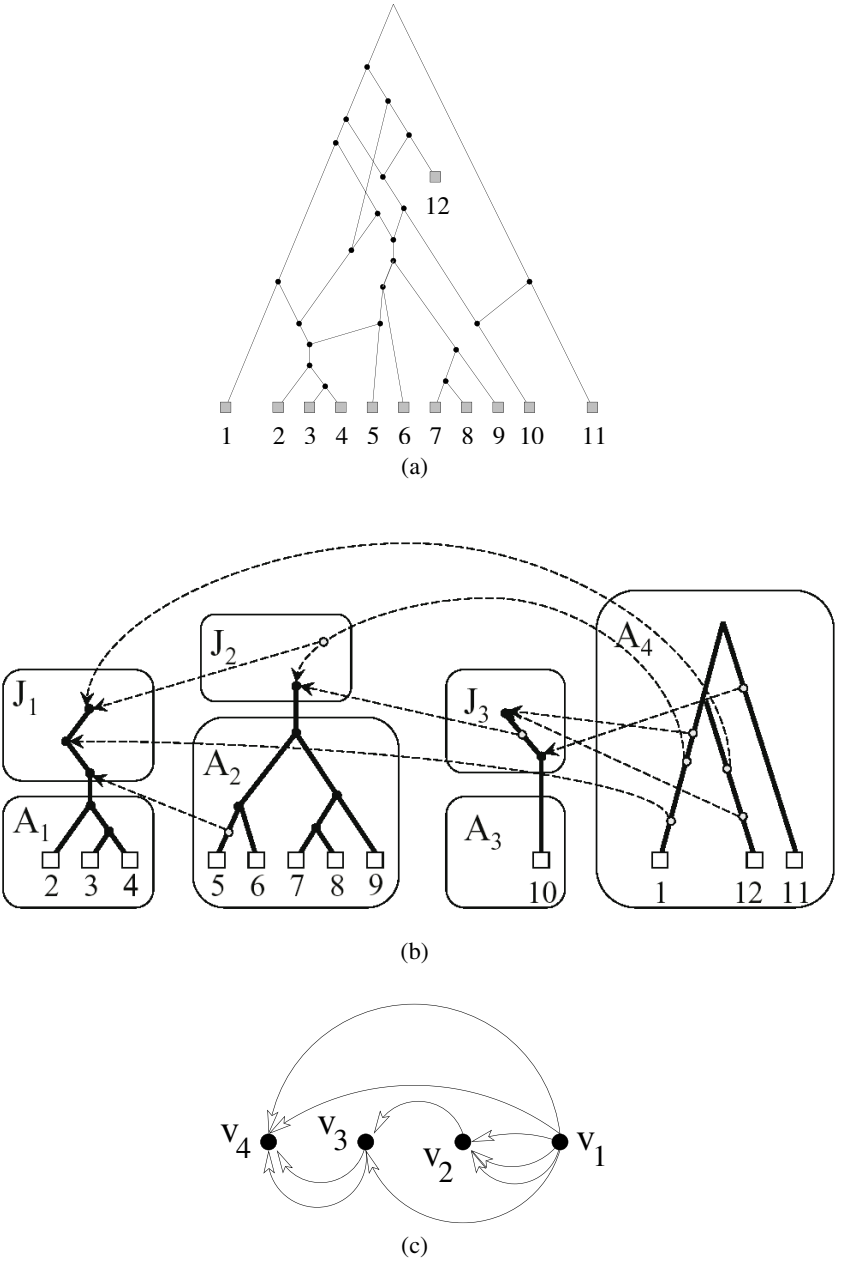


Fig. 2. (a) A phylogenetic network N. (b) A T-decomposition D of N. (c) The dependency digraph $K^{N,D}$.

4. *$\{L(A_i)\}_{1 \leq i \leq m}$, where $L(A_i)$ denotes the leaf set of A_i, forms a partition of $L(N)$.*
5. *If N is binary, then N_i is binary for all $1 \leq i \leq m$.*

Proof.

1. By definition N_m does not have network nodes.
2. Observe that by the algorithm, $E(N_{i+1}), E(A_i), E(J_i)$ and the edge connecting A_i and J_i, form a partition of $E(N_i)$, $1 \leq i \leq m-1$.
3. At each step of the decomposition algorithm, we remove all nodes in the computed connection; N_m does not have network nodes.
4. First notice that $L(A_i)$ is a subset of $L(N_i)$. Assume $L(N_i) - L(N) \neq \emptyset$. Then there exists a node v with outdegree 0 in $L(N_i)$ but not in $L(N)$, which means either $v \in Tr(N)$ or $v \in Nt(N)$. If $v \in Nt(N)$ then all nodes immediately above v except one are in A_i or J_i, which is not possible since v cannot be lower than the lowest network node determining A_i. If $v \in Tr(N)$, then $v \in L(J_i)$, and at least two nodes in J_i are immediately below v, contradicting the fact that N is SH-loop free. Therefore, $L(A_i) \subseteq L(N_i) \subseteq L(N)$. Since A_i is nonempty, it has a lowest node, which must be a leaf in $L(N_i)$. Finally, notice that $L(N_{i+1}) = L(N_i) - L(A_i)$, $1 \leq i \leq m-1$.
5. Straightforward.

Let (u, v) be a terminal edge (i.e., and edge incident with a leaf) that belongs to connection J_i; v is a tree node in N. If N is binary, then for the three edges incident to v, two belong to the same component, because v is suppressed in the i'th step in the decomposition algorithm. We define $\iota(u, v)$ to be the index of this component. It is straightforward to show that $\iota(u, v) > i$.

Finally, we show that exactly one terminal edge from each component in a T-decomposition is used to induce a tree T.

Lemma 3. *If T is a tree induced by a network N, and D is a T-decomposition of N, then exactly one terminal edge from each connection in D is used to induce that tree.*

Proof. Assume the two terminal edges $e_1 = (x_1, y_1)$ and $e_2 = (x_2, y_2)$ from connection J_i are needed to induce tree T. Further, assume v_i is the root of A_i, and S_i is the leaf set of A_i. Assume, as well, that u_i is the network node such that (u_i, v_i) is an edge in N. Notice that each of the two edges e_1 and e_2 were either a single edge or a path of edges in N.

Exactly one of the two edges, say e_1, reaches S_i in T, whereas the other edge, e_2, reaches a set S' of leaves, where $S_i \cap S' = \emptyset$; otherwise, the underlying undirected graph of T contains a cycle – a contradiction.

It follows that the path p from x_2 to u_i contains a node z, dividing p into two paths p_1 (from x_2 to z) and p_2 (from z to u_i), and such that there is a terminal edge (z, w) in some connection J_j, $i \neq j$, where the set S' of leaves is under w. The node w must be a network node.

Now consider all possible nodes on the path p_2 (between node z and node u_i, exclusive). If there were no such nodes, and since u_i is a network node, it follows that node z has two network node children (w and u_i) – a contradiction to the assumption that N

does not have a tree node whose two children network nodes (notice that node z cannot be a network node, since by definition, a network node has outdegree 1).

Now, assume there were nodes on the path p_2 from z to u_i, and let s be such a node. If s were a tree node, then there exists at least one leaf t in N that is reachable from s through paths that do not contain any network nodes (otherwise, the subnetwork rooted at s contains a tree node whose two children are also network nodes, which is a contradiction). In this case, the edges on the path from x_2 to s cannot be in the connection J_i (those edges would be in maximal clade A_m) – a contradiction to the assumption that edge e_2 is a terminal edge in connection J_i. Therefore, all nodes on the path p_2 from z to u_i are network nodes, and hence tree node z has two children that are network nodes – a contradiction.

Therefore, exactly one terminal edge from each connection is used to induce a tree T in a network N.

4.4 Dependency Graphs

Given a network N and its T-decomposition D, we define the *dependency digraph* $K^{N,D}$ to facilitate our algorithm design.

Definition 3. *Given a network N and its T-decomposition $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$, the dependency graph is a directed multigraph $K^{N,D}$, where node v_i in $K^{N,D}$ corresponds to the pair (A_i, J_i) in D, and edge (v_i, v_j) $(i > j)$ in $K^{N,D}$ corresponds to a terminal edge connecting J_j and J_i in N.*

Figure 2(c) shows a dependency graph of the network and T-decomposition given in Figures 2(a) and 2(b). In other words, $K^{N,D}$ is the graph resulting from replacing each component (A_i, J_i) in D by a single node v_i, and hence, $K^{N,D}$ is necessarily connected. If $K^{N,D}$ had a cycle, then N would be cyclic. Therefore, we have the following result.

Proposition 5. *The dependency graph $K^{N,D}$ is connected and acyclic. Moreover, (v_i, v_j) is an edge in $K^{N,D}$ only if $i > j$.*

At the end of the decomposition process, we keep a matrix that shows which component each edge belongs to. So querying which component an edge (u, v) belongs to, as well as computing the value of $\iota(u, v)$, take $O(1)$ time. Thus, computing the dependency graph $K^{N,D}$ takes $O(m|E(N)|) = O(|Nt(N)||V(N)|)$ time. Further, in $K^{N,D}$, we keep track of the correspondence between edges of N and edges of $K^{N,D}$.

5 Reduced Inheritance Profiles and the Cluster Lemma

Given a T-decomposition D of cardinality m, a *reduced inheritance profile* is a set of size m that contains exactly one terminal edge per connection in the decomposition. We only keep the terminal edges because all inheritance profiles having the same set of terminal edges necessarily induce the same tree. A reduced inheritance profile extends into an inheritance profile in a straightforward manner, as no edges in the reduced inheritance profile are incident with the same network node. We say that a reduced inheritance profile is *valid* if it induces a tree. The following results show the correspondence between inheritance profiles and reduced inheritance profiles.

Proposition 6. *Let D be a T-decomposition of a network N. Then,*

1. *For each valid reduced inheritance profile IP there exists a valid inheritance profile IP' that contains IP and induces the same tree.*
2. *For each valid inheritance profile IP' there exists a unique valid reduced inheritance profile IP that induces the same tree.*

Proof.

1. To compute the inheritance profile IP', for each connection J_i we take the unique path from the terminal edge e_i to the root of J_i; for each network node v_i in J_i, we choose the edge on the path as the value of x_i in the inheritance profile. For other nodes we make choices arbitrarily. Since no leaf can be reached from the root through nodes not on the path, choosing different edges incident with these nodes in the profile do not affect the tree topology.
2. Given a valid inheritance profile IP', for each connection J in the D there is exactly one path connecting a terminal edge in J and the root of J. We retain this edge and drop all other terminal edges in $J \cap IP'$. We obtain IP by repeating the same process for all connections.

The dependency graph can be seen as a compact representation, mainly for reduced inheritance profiles.

Lemma 4. *Let D be a T-decomposition of a network N, $K^{N,D}$ be the dependency graph, and IP be a valid reduced inheritance profile. Then, $K^{N,D}$, restricted to the edges in IP, forms a tree.*

We are now in position to show the correlation between clusters and a T-decomposition of a network – a result that forms the basis for our algorithms.

Lemma 5. (Cluster Lemma) *Let $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$ be a T-decomposition of a network N. Each cluster C induced by N can be written as $C = \cup_j C_j$, where each C_j is an element of $\{L(A_i) : 1 \leq i \leq m\}$, except for at most one of the C_j's, which may be a proper subset of an element of $\{L(A_i) : 1 \leq i \leq m\}$.*

Proof. The lemma is trivially true for $|C| = 1$. Assume $|C| > 1$, and let T be a phylogenetic tree that contains C. Let IP be a reduced inheritance profile that induces T in N. Construct the tree T' in the dependency graph according to Lemma 4. Let v in N be a lowest common ancestor of all leaves in C. Node v must be a tree node (if it were a network node, then the node below v would also be a common ancestor, contradicting the fact that v is a lowest common ancestor). Let $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$ be k maximal clades in D and such that each of them has nonempty intersection with C. Let e_{i_j} be the terminal edge in J_{i_j} that is an element of IP. There are two cases: (1) v is in a maximal clade A_l but not in any connection in D. Let $L(v)$ be the cluster in A_l below v; or (2) v is in a connection J_q in D. Then there is a terminal edge $(u, v) \in J_q$. Let $l = \iota(u, v)$, and let $L(v)$ be the cluster in A_l below v. In both cases, $L(v)$ is nonempty, and if $L(v) \neq L(A_l)$, then $L(v)$ is the C_j that is a proper subset of an element of $\{L(A_i) : 1 \leq i \leq m\}$. Furthermore, for any A_{i_j}, $i_j \neq l$, $1 \leq j \leq k$, $i_j \neq m$, any path from v to a leaf in $L(A_{i_j})$ passes through e_{i_j} by Lemma 6; thus, $L(A_{i_j}) \subseteq C$. Any leaf

in $L(A_{i_j}) \setminus L(A_x)$ can be reached from v through at least one terminal edge in IP, so by Lemma 5, $l = \max\{i_1, \ldots, i_k\}$.

Corollary 1. *Let $D = \{(A_i, J_i)\}_{1 \leq i \leq m}$ be a T-decomposition of a network N, IP be a reduced inheritance profile, and C be a cluster. Then, when restricted to the nodes whose corresponding maximal clades have nonempty intersection with C and to the edges in IP, the dependency graph $K^{N,D}$ forms a tree. Further, the root of that tree has the highest index.*

Proof. Using the collapsing argument in the proof of Lemma 4, and Proposition 5.

Corollary 1 gives an algorithm for computing the component that contains the node which "determines" a cluster C: compute the corresponding subtree in the dependency graph and return the component corresponding to the root. If one of the C_j's in the Cluster Lemma is a proper subset of an element in $\{L(A_i) : 1 \leq i \leq m\}$, the component that contains that C_j must be the root, based on the proof of the Cluster Lemma.

6 Polynomial-Time Algorithms for the Decision Problems

6.1 Deciding the Network-Cluster Containment Problem

We are finally in a position to describe a polynomial-time algorithm for deciding the Network-Cluster Containment Problem. The algorithm is given in Figure 3. Let $D = \{A_i, J_i\}_{1 \leq i \leq m}$ be a T-decomposition of a network N, and let $C \subset L(N)$ be a cluster. We define the set $\psi(C) = \{i : 1 \leq i \leq m \text{ and } L(A_i) \subseteq C\}$. The basic idea is to compute a set E_C of edges that are *incompatible* with C, i.e., edges that cannot co-exist with C in the same tree induced by N.

Algorithm TestCinN(N,C)

1. Compute a T-decomposition $D = ((A_1, J_1), \ldots, (A_m, J_m = \emptyset))$.
2. Test if C can be decomposed into the following form: $\bigcup_{i \in \psi(C)} L(A_i) \cup L'$, where $L' = \emptyset$ or $L' \subset L(A_l)$ for some l. If not, return NO. If $L' = \emptyset$ then let $l = \max_{i \in \psi(C)} i$.
3. Partition $V = V(K^{N,D})$ into two sets: $V_C = \{v_i | i \in \psi(C)\}$ and $\overline{V_C} = V - V_C$. Compute the set $E_C = \{e_{ij} = (v_i, v_j) | e_{ij} \in E(K^{N,D}), v_i \in \overline{V_C}, v_j \in V_C, j \neq l\}$.
4. If $L' \neq \emptyset$, test if L' is a cluster of A_l. If not, return NO; otherwise:
   (a) Let v' be the root of the clade whose leaf set is L'.
   (b) For each terminal edge (u, v) in A_i, for some $i \in \psi(C)$ and $\iota(u, v) = l$, (edge (u, v) connects the i'th component to the l'th component), add (u, v) to E_C if u is not a descendant of v' in N_i.
5. Remove all terminal edges in N and that correspond to edges in E_C (and apply forced contraction operations); let the result be N_C. If N_C is connected, return YES. Otherwise, return NO.

Fig. 3. Algorithm **TestCinN** for deciding the NETWORK-CLUSTER CONTAINMENT PROBLEM

Theorem 2. *Algorithm TestCinN(N,C) decides the Network-Cluster Containment Problem in $O(|V(N)|^2)$ time.*

Proof. We first show the correctness of the algorithm. Step 3 in the algorithm is well defined: if $(v_i, v_j) \in E(N)$ then $i > j$ by Proposition 5.

Assume N_C is connected. It is easy to show N_C is still a network (note that we only remove terminal edges). By Proposition 2, N_C induces a tree T'; note that N also induces T'. There exists a reduced inheritance profile IP that induces T' in N_C. Let e be the terminal edge in $IP \cap E(J_l)$. We now show T' contains C. For any terminal edge e', assume it is from component i. Assume $i \in \psi(C)$. If $e' \in IP$, then e' is below e in T', otherwise e' is in E_C. So the cluster below e in T contains each $L(A_i)$, $i \in \psi(C)$. Now assume $i \notin \psi(C)$ and $i \neq l$. If $e' \in IP$, then $\iota(e') \notin \psi(C) \cup \{l\}$. Therefore $\bigcup_{i \in \psi(C)} L(A_i) \subseteq C$, and for each $i \notin \psi(C) \cup \{l\}$, $L(A_i) \cap C = \emptyset$. Finally, if L' is not empty, notice that for any terminal edge e' from component i, $i \in \psi(C)$, if $e' \in IP$, then e' is lower than v' in T'. Therefore the cluster determined by v' in T' is exactly $\bigcup_{i \in \psi(C)} L(A_i) \cup L'$.

Now, assume N induces tree T that contains cluster C; we show that N_C is connected. Let $G = K^{N,D}$ be the dependency graph, and let G_C be the graph obtained by removing edges in E_C from $K^{N,D}$. Let IP be a reduced inheritance profile that induces T. If none of the edges in T is in E_C then G_C (and hence N_C) is connected. To see this is true, assume otherwise; then there exists $(u, v) \in IP \cap E_C$. Either (u, v) is an edge in step 3 or step 4(b). The first case contradicts Corollary 1, since the edge connects some vertex v_i, $i \in \psi$ in G_C to a vertex v_j above v_l. In the second case, the cluster determined by u in A_l properly contains L'. If $(u, v) \in J_i$ (in which case $L(A_i) \subseteq C$), then for any vertex below u in T', its corresponding cluster does not contain $L(A_i)$.

We now analyze the running time of the algorithm. (Recall that N is binary.) Step 1 takes $O(|Nt(N)||V(N)|)$ time. Steps 2 and 3 take $O(|L(N)|)$ time if we keep track of which component each leaf in $L(N)$ belongs to, when we compute D. In step 4, first notice the number of terminal edges is bounded by $2|Nt(N)|$; for each terminal edge, testing its membership in E_C takes constant time. In Step 5, testing if L' is a cluster in A_i takes $O(|L(A_i)|) = O(|L(N)|)$ time by doing a depth-first search; we can also find v in 5(a) at the same time. Testing for each (u, v) if it should be added to E_C takes constant time, and there are $O(|Nt(N)|)$ of them. Finally, in Step 6, removing E_C from N takes $O(|E_C|) = O(|Nt(N)|)$ time; testing the connectedness of N_C can be achieved by a depth-first search. The overall running time is $O(|Nt(N)||V(N)|)$=$O(|V(N)|^2)$. If the T-decomposition is given, the running time is $O(|V(N)| + |E(N)|) = O(|V(N)|)$.

Based on the proof of Theorem 2, we have the following result.

Corollary 2. *Given any network N, a phylogenetic tree T, and a cluster C in T, N induces T if and only if N_C induces T.*

Proof. Let IP be an inheritance profile that induces T. Notice that no edge of IP is in E_C. Since IP induces T in N, it also induces T in N_C.

6.2 Deciding the Network-Tree Containment Problem

Using algorithm TestCinN(N,C), Figure 4 describes our polynomial-time algorithm for deciding the Network-Tree Containment Problem.

Algorithm TestTinN(N,T)

1. Compute a T-decomposition $D = ((A_1, J_1), \ldots, (A_m, J_m = \emptyset))$.
2. For each nontrivial cluster C in T ($C \neq L(N)$ and $|C| > 1$), call TestCinN(N,C); update N by removing E_C from N.
3. If N is connected, return YES; otherwise, return NO.

Fig. 4. Algorithm **TestTinN** for deciding the NETWORK-TREE CONTAINMENT PROBLEM

Theorem 3. *Algorithm TestTinN(N,T) decides the Network-Tree Containment Problem in $O(|V(N)||L(N)|)$ time.*

Proof. We denote by N' the network obtained at the end of Step 2. We want to show that N induces T if and only if N' is connected. By Corollary 2, after each iteration in Step 2 of the algorithm, the new N still induces T; so N' is connected.

Assume N' is connected. Then N' induces a tree T'. Let IP be an inheritance profile in N' that induces T'. It suffices to show $T' = T$ since N' is a subnetwork of N. Now consider any nontrivial cluster C in T. C can be decomposed using the Cluster Lemma. Let $l = \max \psi(C)$, and let $e = P \cap E(J_l)$. Since we call TestCinN() with C as the input cluster, every leaf in C in T' is below e. If L' in the TestCinN algorithm is empty, the cluster below e in T' is C. If L' is not empty, L' is a cluster in A_l; in this case let v' be the lowest common ancestor in L'. The cluster in T' determined by v' is C.

We now analyze the running time of the algorithm. (Recall that N is binary.) We only need to compute T-decomposition once, which takes $O(|Nt(N)||V(N)|)$ time. Each iteration in Step 2 takes $O(|V(N)|)$ time, and the number of clusters in T is $O(|L(N)|)$. The final step takes $O(|V(N)| + |E(N)|) = O(|V(N)|)$ time. The overall time is therefore $O(|V(N)|^2)$.

7 Conclusion and Future work

Phylogenetic networks are the appropriate model for evolutionary histories in the presence of reticulation events. Very little is known about their combinatorial properties, and many problems are still open in this domain. In this paper, we presented polynomial-time algorithms for two major problems, namely (1) deciding whether a tree is induced by a network, and (2) deciding whether a cluster is induced by a network. Those two algorithms are based on a novel network decomposition that we introduced. Directions for future research include enumerating the numbers of trees and clusters induced by a network, efficient techniques for network space traversal, and accurate reconstruction of networks from sets of clusters and trees.

References

[1] D. Bryant and V. Moulton. NeighborNet: An agglomerative method for the construction of planar phylogenetic networks. In R. Guigo and D. Gusfield, editors, *Proc. 2nd Workshop Algorithms in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pages 375–391. Springer Verlag, 2002.

[2] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, MA, 2003.

[3] D.H. Huson. SplitsTree: A program for analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[4] W.P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

[5] B.M.E. Moret, L. Nakhleh, T. Warnow, C.R. Linder, A. Tholse, A. Padolina, J. Sun, and R. Timme. Phylogenetic networks: modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004.

[6] L. Nakhleh, T. Warnow, and C.R. Linder. Reconstructing reticulate evolution in species – theory and practice. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004)*, pages 337–346, 2004.

[7] D.L. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis. Phylogenetic inference. In D.M. Hillis, B.K. Mable, and C. Moritz, editors, *Molecular Systematics*, pages 407–514. Sinauer Assoc., Sunderland, Mass., 1996.

Minimum Parent-Offspring Recombination Haplotype Inference in Pedigrees

Qiangfeng Zhang[1], Francis Y.L. Chin[2], and Hong Shen[3]

[1] Department of Computer Science,
University of Science and Technology of China, Hefei 230026, China
qfzhang@mail.ustc.edu.cn
[2] Department of Computer Science, The University of Hong Kong,
Pokfulam, Hong Kong
chin@cs.hku.hk
[3] Graduate School of Information Science, JAIST, Ishikawa, Japan
shen@jaist.ac.jp

Abstract. The problem of haplotype inference under the Mendelian law of inheritance on pedigree genotype data is studied. The minimum recombination principle states that genetic recombinations are rare and haplotypes with fewer recombinations are more likely to exist. Given genotype data on a pedigree, the problem of Minimum Recombination Haplotype Inference (MRHI) is to find a set of haplotype configurations consistent with the genotype data having the minimum number of recombinations. In this paper, we focus on a variation of the MRHI problem that gives more realistic solutions, namely the k-MRHI problem, which has the additional constraint that the number of recombinations in each parent-offspring pair is at most k. Although the k-MRHI problem is NP-hard even for $k = 1$, the k-MRHI problem with $k > 1$ can be solved efficiently by dynamic programming in $O(nm_0^{3k+1}2^{m_0})$ time by adopting an approach similar to the one used by Doi, Li and Jiang [4] on pedigrees with n nodes and at most m_0 heterozygous loci in each node. In particular, the 1-MRHI problem can be solved in $O(nm_0^4 2^{m_0})$ time. We propose an $O(n^2 m_0)$ algorithm to find a node as the root of the pedigree tree so as to further reduce the time complexity to $O(m_0 min(t_R))$, where t_R is the number of feasible haplotype configuration combinations in all trios in the pedigree tree when R is the root. If the pedigree has few generations, the 1-MRHI problem can be solved in $O(min\{nm_0^4 2^{m_0}, nm_0^{l+1}2^{\mu_R+l}\})$ time, where μ_R is the number of heterozygous loci in the root, and l is the maximum path length from the root to the leaves in the pedigree tree. Experiments on both real and simulated data confirm the out-performance of our algorithm when compared with other popular algorithms. In most real cases, our algorithm gives the same haplotyping results but runs much faster. In some real cases, other algorithms give an answer which has the least number of recombinations, while our algorithm gives a more credible solution and runs faster.

1 Introduction

The modeling of human genetic variation is critical to the understanding of the genetic basis for complex diseases. *Single nucleotide polymorphisms* (*SNPs* [13])

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 100–112, 2005.

are the most frequent form of this variation. The Human Genome Project and other large-scale efforts have identified millions of SNP markers that can be used in genetic studies. Although each marker can be analyzed independently, it is much more informative to analyze them in groups. Therefore, it is useful to analyze *haplotypes* (*hap*loid geno*types*), which are sequences of linked markers on a single chromosome. In diploid organisms, such as humans, chromosomes come in pairs, and experiments often yield *genotype* information, which blend haplotype information for chromosome pairs. There is growing evidence that, in order to better characterize the role of a candidate gene, full haplotype information should be exploited instead of using only genotype information. Unfortunately, it is both time-consuming and expensive to derive haplotype information experimentally. This explains the increasing interest in inferring haplotype information, or *haplotyping*, computationally [2][6].

Input genotype data can be with or without any other *pedigree* information. Haplotyping pedigree data is believed to be more reliable than haplotyping population data for unrelated individuals: the constraint provided by parents-offspring relationships in a pedigree could force one to settle on a unique haplotype configuration as being most probable.

Genetic research shows that recombinations are rare in human data [5]. The genomic DNA can be partitioned into long blocks such that recombinations within each block are rare or even nonexistent. Thus it is believed that haplotype configurations with fewer recombinations should be preferred in haplotype inference [11][12].

The *Minimum-Recombination Haplotype Inference* (*MRHI*) problem, which is NP-hard [4], is to find a haplotype configuration with minimum number of recombinations for a given pedigree genotype data. Various algorithms have been presented for the MRHI problem [8][7][12][15]. In some cases, however, the MRHI model might yield unrealistic results in which a few parent-offspring pairs have many recombinations while others have no or few recombinations. We present a more realistic problem, called the k-MRHI problem which basically is the MRHI problem, but with an additional constraint that the number of recombinations in each parent-offspring pair is bounded by a constant k. The k-MRHI problem is NP-hard even for $k = 1$.

The k-MRHI problem can be solved by a dynamic programming (DP) algorithm which is very similar to the algorithm by Doi, Li and Jiang [4]. By avoiding studying all 2^{3m_0} haplotype configurations in each parents-offspring trio, our algorithm takes $O(nm_0^4 2^{m_0})$ time when $k = 1$, instead of the $O(nm_0 2^{3m_0})$ time needed by [4] for the MRHI problem on pedigrees with n nodes and at most m_0 heterozygous loci in each node. Note that not all nodes have m_0 heterozygous loci, and the number of feasible haplotype configurations at a node is limited by the number of feasible haplotype configurations of its neighbors, and thus the number of possible haplotype configurations at a node can be much less than 2^{m_0}. This observation leads to the idea of choosing different nodes in the pedigree as the root of the tree in speeding up the algorithm. The main contributions of this paper are: (1) to define a more realistic problem for haplotype inference

(k-MRHI), (2) to give a more efficient and practical DP algorithm for the haplotype inference problem with improved time complexities, and (3) to present an efficient algorithm to find the root in the pedigree for better performance in the DP algorithm.

2 Preliminaries

Haplotypes and genotypes consist of linked *genetic markers* which are small segments of DNA with some specific features. The physical position of a marker on a chromosome is called a *locus* and its state is called an *allele*. Without loss of generality, the two alleles of a biallelic (2-state) SNP can be denoted by '0' and '1', and a haplotype h with m loci is presented as a string of length m over $\{0,1\}^m$, and a genotype g as a string over $\{0,1,2\}^m$. Haplotype pair $\langle h_1, h_2 \rangle$ is compatible with a genotype g if (a) the two alleles of h_1 and h_2 are the same at the same locus, for example '0' (respectively '1'), then the corresponding locus of g should also be '0' (respectively '1'), which denotes a *homozygous* site; otherwise, (b) the two alleles of h_1 and h_2 are different, then the corresponding site of g should be '2', which denotes a *heterozygous* site.

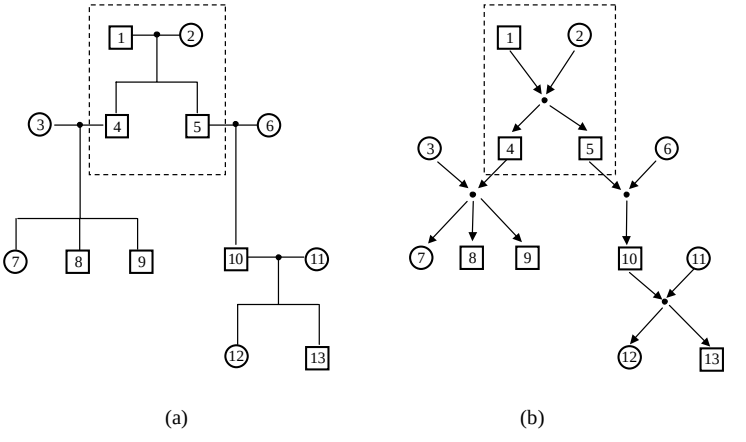


Fig. 1. The pictorial representation and graph representation of a pedigree

A pedigree is a fundamental structure used in genetics. Figure 1(a) shows the pictorial representation (used by the biologists) of a pedigree with 13 nodes. A square represents a male node, a circle represents a female node, and a black dot represents a mating node. The subgraph in the dashed square is a typical nuclear family, which contains a father (node 1), a mother (node 2) and two children (nodes 4 and 5). The children are placed under their parents. Nodes 1, 2 and 4 consist a parents-offspring trio, nodes 1 and 4, nodes 1 and 5, nodes 2

and 4, nodes 2 and 5 are parent-offspring pairs. We define a pedigree formally as in [4].

Defintion 1[4]**.** *A* **pedigree** *is a weakly connected directed acyclic graph* $P = (V, E)$*, where* $V = M \cup F \cup N$*, with* M *stands for the male nodes,* F *the female nodes,* N *the mating nodes, and* $E = \{(u, v)\}$ *with* $u \in M \cup F$ *and* $v \in N$*, alternatively* $u \in N$ *and* $v \in M \cup F$*.*

Figure 1(b) shows the graph representation of the pedigree given in Figure 1(a). A sub-graph containing the father, the mother, and their children is a *nuclear family*. A nuclear family can also be represented by a mating node which connects them together. A *parents-offspring trio*, or just *trio*, consists of two parents and one of their children; and a *parent-offspring pair* (*PO-pair*) refers to a father and his child or a mother and her child. In this paper, we assume that the pedigree never forms a cycle if the directions of edges are ignored (no *mating-loop*).

Each individual node in a pedigree is associated with its genotype. In the absence of genetic mutation, at each locus, the child must inherit one allele from its father and the other from its mother. This is known as the *Mendelian law of inheritance*. Usually, one haplotype of a child is inherited as a whole from one of the two haplotypes of a parent. However, *recombinations* may occur, where the two haplotypes of a parent get shuffled due to a crossover of a chromosome and one of the shuffled copies (*recombinant*) is passed on to the child. However, genetic research shows that recombinations are rare in human genetics. Thus we are interested in finding the haplotype configurations such that the total number of recombinations in the whole pedigree is minimized.

Defintion 2[12]**. Minimum Recombinant Haplotype Inference (MRHI) Problem:** *Given a pedigree graph P, each individual node of P associates with a genotype. Find a haplotype configuration for the pedigree that each haplotype pair at each node is an explanation of its corresponding genoytype and the total number of recombinations is minimized.*

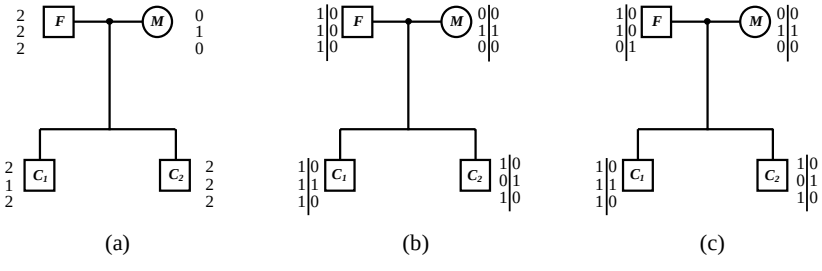


Fig. 2. Two different solutions for a pedigree of a nuclear family

Figure 2(a) shows a pedigree of a nuclear family containing a father F, a mother M and 2 children C_1, C_2. Figure 2(b) gives a solution with no recom-

bination in trio (F, M, C_1) and 2 recombinations in trio (F, M, C_2). Figure 2(c) gives another solution, which also has 2 recombinations in total, but at most one recombination in each trio, i.e. at most one recombination in each PO-pair. As genetic studies show that recombinations are rare to have 2 recombinations within one PO-pair, *e.g.*, there are 13% single recombinations versus 0.84% double recombinations in the Drosophila autosomal genes [5], Figure 2(c) should be a more credible solution than Figure 2(b) for the haplotype inference problem.

Defintion 3 . k-Recombination Haplotype Inference (k-MRHI) Problem: *Given a pedigree graph P with each individual node associated with a genotype, find a haplotype configuration that is compatible with the genotypes at all nodes having the minimum number of recombinations and no more than k recombinations in each PO-pair.*

3 A Dynamic Programming Algorithm for k-MRHI

3.1 The 1-MRHI Problem ($k = 1$)

In [4], Doi *et al.* gives a proof for the NP-hardness of the MRHI problem by a reduction from MAX CUT. In their construction, the number of recombinations within each PO-pair is limited to 1. This trivially implies that the k-MRHI problem, even for $k = 1$, is also NP-hard.

However, in most cases, we can find a feasible solution for a k-MRHI instance with $k < 2$. As we have mentioned before, more than 1 recombination within a PO-pair is very unlikely in reality. Therefore, we shall focus on the 1-MRHI problem first and generalize to the k-MRHI problem later.

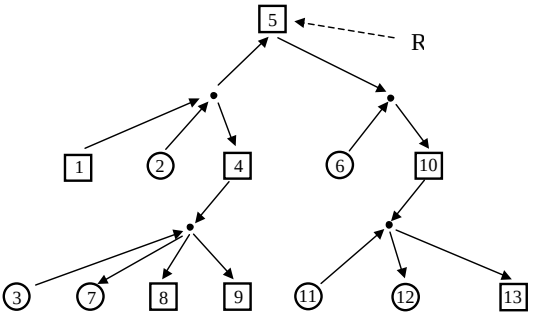


Fig. 3. The searching tree of the pedigree in Figure 1

A locus-based dynamic programming (DP) algorithm for the k-MRHI problem was presented in [4], with a time complexity of $O(nm_0 2^{3m_0})$, where m_0 is the maximum number of heterozygous loci in the genotype at each node of a loopless pedigree. We adopt a similar DP approach to solve the 1-MRHI problem

by (1) assigning an arbitrary node R in the pedigree as the root (an example is given in Figure 3, which shows a rooted tree at node 5 for the pedigree in Figure 1); (2) recursively finding $num[R][s]$, the minimum number of recombinations required for all feasible haplotype configurations s of R; and (3) selecting the haplotype with the minimum number of recombinations as the solution.

Let $num[r][s]$ denote the minimum number of recombinations required in the sub-tree rooted at r with the haplotype configuration s under the constraint that there is at most 1 recombination in each PO-pair of the sub-tree. If r has multiple mating nodes as its tree sons, we compute each mating node separately. Each child mating node of r defines a unique nuclear family, which may contain r as a parent or a child and the computation of $num[r][s]$ is performed recursively in that nuclear family.

Suppose that the nuclear family consists of father F, mother M and children $C_1, \cdots, C_d$. If r is a leaf node, $num[r][s] = 0$ for any of haplotype configuration s; else, if r is M (or F, respectively) with haplotype configuration s, then:

$$num[r][s] = \min_{p}(num[F][p] + \sum_{i=1}^{d} \min_{c_i}(num[C_i][c_i] + numtrio(p, s, c_i))) \quad (1)$$

where p denotes the haplotype configuration at node F and c_i the haplotype configuration at C_i, one of the d children in this nuclear family. numtrio (p, s, c_i) returns the minimum number of recombinations required for a trio consisting of F, M, and C_i with the haplotype configurations p, s and c_i respectively, under the constraint that no PO-pair can have more than one recombination. If there does not have any feasible solution, then numtrio (p, s, c_i) will return ∞, which indicates "no solution".

Similarly, if r is C_j with haplotype configuration s, then we have:

$$num[r][s] = \min_{p,q}(numtrio[p, q, s] + num[F][p] + num[M][q] + \sum_{i=1, i\neq j}^{d} \min_{c_i}(num[C_i][c_i] + numtrio(p, q, c_i))) \quad where\ r = C_j \quad (2)$$

where p, q and c_i are defined as before for haplotype configurations at F, M and C_i respectively.

Note that the above algorithm is the same as that presented in [4], and thus would have the same time complexity. However, a reduction in time complexity is possible from an important observation: it is not necessary to consider all combinations of haplotype configurations in each trio, which number $O(2^{3m_0})$ in total, because many combinations of haplotype configurations will be infeasible, *i.e.* will not have at most one recombination per PO-pair.

For example, assume the genotype of F is $(2, 2, \cdots, 2)$ of length m_0 and with haplotype configuration $s = \langle h_{s1}, h_{s2}\rangle$ and h_{c1} in the haplotype $c = \langle h_{c1}, h_{c2}\rangle$ of C_i is inherited from s with no more than 1 recombination. There are $m_0 + 1$ ways of forming h_{c1} by inheriting its first w alleles from the first w alleles in h_{s1} and the remaining $(m_0 - w)$ alleles from h_{s2} with $0 \leq w \leq m_0$. Similarly, there

are another $m_0 + 1$ ways of forming h_{c1} from the first w alleles in h_{s2} and the remaining $(m_0 - w)$ alleles from h_{s1} . Since there are double-counting in these two cases when $w = 0$ and m_0, the number of feasible haplotype configurations of c is limited to $2m_0$, and the time complexity of the algorithm can be much reduced if we limit the number of configurations needed to be searched for the optimal result. More precisely, suppose r in Equation (1) is M (or F), and $s = \langle h_{s1}, h_{s2} \rangle$, let N_s be the set of feasible haplotype configurations $c = \langle h_{c1}, h_{c2} \rangle$ that can be inherited by child C_i from s of r with no more than one recombination. Thus, $|N_s| \leq 2m_0$. As h_{c2} is inherited from the haplotype configuration $q = \langle h_{q1}, h_{q2} \rangle$ of F, let N_c be the set of feasible haplotype configurations of F which can produce the haplotype configuration c in C with no more than one recombination. Let $N'_{s,C_i} = \cup_{c \in N_s} N_c$, which indicates the set of feasible haplotype configurations at F which can go together with haplotype s at M to produce children C_i with no more than one recombination in the father-child pair and in the mother-child pair. Obviously, $N'_{s,C_i} \leq 4m_0^2$.

As each haplotype configuration of F should be able to produce any of the children $C_1, \cdots, C_d$, the set of feasible haplotype configurations in F is $N'_s = \cap_i N'_{s,C_i}$. Equation (1) can be rewritten as:

$$num[r][s] = \min_{p \in N'_s} (num[F][p] + \sum_i \min_{c_i \in N_s} (num[C_i][c_i] + numtrio(p, s, c_i))) \quad (3)$$

As for Equation (2), if r is C_j and its haplotype configuration $s = \langle h_{s1}, h_{s2} \rangle$, let $N_{s,F}$ and $N_{s,M}$ be the sets of feasible haplotype configurations in F and M, which can produce C_j with haplotype configuration s. As $|N_{s,F}| \leq 2m_0$ and $|N_{s,M}| \leq 2m_0$, let $N_{p,C_i}(N_{q,C_i})$ be the set of feasible haplotype configurations on another child C_i with haplotype configuration p in F (q in M) and $N''_{p,q} = N_{p,C_i} \cap N_{q,C_i}$ be the set of feasible haplotype configurations for each child C_i which can concurrently appear with the haplotype configuration s of child C_j. Note that $N''_{p,q} \leq 2m_0$ and Equation (2) can be rewritten as:

$$num[r][s] = \min_{p \in N_{s,F}, q \in N_{s,M}} (numtrio(p, q, s) + num[F][p] + num[M][q] + \sum_{i \neq j} \min_{c_i \in N''_{p,q}} (num[C_i][c_i] + numtrio(p, q, c_i))) \quad where\ r = C_j \quad (4)$$

Theorem 1. *The above dynamic programming algorithm can solve the 1-MRHI problem in $O(nm_0^4 2^{m_0})$ time and $O(n2^{m_0})$ space for pedigree with n nodes and at most m_0 heterozygous loci in each node.*

Proof. The rooted tree can be constructed in $O(n)$ time. As we have to consider the $8m_0^3$ combinations in each trio for each haplotype configuration of a node and we need $O(m_0)$ time to compute *numtrio* for each haplotype configuration combination in a trio, it may take $O(m_0^4 2^{m_0})$ time to process each trio. There are at most n parent-offspring trios in the pedigree, so the time complexity is $O(nm_0^4 2^{m_0})$. Furthermore, we need to store the array *num* and pointers for backtracking. The size of *num* is $O(n2^{m_0})$, so is the number of pointers. Thus the space complexity is $O(n2^{m_0})$.

3.2 The k-MRHI Problem

We have argued that in most cases, feasible solutions exist for 1-MRHI. However, there are still some instances that require more recombinations within each PO-pair. In almost all practical cases, there are at most 2 recombinations within each PO-pair. In the following, we generalize the DP algorithm to the general k-MRHI problem with some modifications.

We need to modify the definition of neighboring haplotype configurations set from N_s to $N_s^{(k)}$: for each haplotype configuration $c = \langle h_{c1}, h_{c2}\rangle \in N_s^{(k)}$, one of $\langle h_{c1}, h_{c2}\rangle$ is inherited from one of $\langle h_{s1}, h_{s2}\rangle$ with no more than k recombinations. So we have $|N_s^{(k)}| = O(m_0^k)$.

Similarly, we modify the definition of $N_s^{'(k)} = \cap_i N_{s,C_i}^{'(k)}$ with N'_{s,C_i} to $N_{s,C_i}^{'(k)}$ in Equation (3) and the definition of $N_{s,F}$ and $N_{s,M}$ to $N_{s,F}^{(k)}$ and $N_{s,M}^{(k)}$, N_{p,C_i} to $N_{p,C_i}^{(k)}$, and $N''_{p,q}$ to $N_{p,q}^{''(k)}$ in Equation (4). Then we have:

Theorem 2. *The time complexity of the DP algorithm solving the k-MRHI problem is $O(nm_0^{3k+1}2^{m_0})$ for pedigree with n nodes and at most m_0 heterozygous loci in each node.*

4 Root Selection for Better Performance

We have shown in Section 3 that in the 1-MRHI problem, the number of feasible haplotype configuration combinations in each trio is no more than $O(m_0^2 2^{m_0})$. However, in practice the feasible haplotype configuration combinations in each trio may be much less than that because of the following reasons: (1) not all nodes have m_0 heterozygous loci; and (2) the number of feasible haplotype configurations a_v of a node v is also bounded by the number of feasible haplotype configurations a_{v_r} of v's neighbor v_r, which can participate in the feasible haplotype configuration combinations in a trio, *i.e.*, $a_v \leq 2\mu_v a_{v_r}$, where μ_v is the number of heterozygous loci in v. Thus, different selections of a node in the pedigree as the root for the DP algorithm will give different processing times. The following we shall discuss an algorithm to find the best root based on the estimated number of feasible haplotype configurations in each node.

Starting from any node R, as root and assuming α_R be the number of feasible haplotypes configurations of R, *i.e.*, $\alpha_R = 2^{\mu_R}$, we will traverse the tree in pre-order and, for each node v, evaluate the number of the feasible haplotype configurations for its neighboring nodes.

If v has multiple mating nodes as v's children, we compute each mating node separately. Each mating node as v's child defines a unique nuclear family, which may contain v as a parent or a child. Suppose that the nuclear family consists of father F, mother M and children $C_1, \cdots, C_k$.

If v is M (or F, respectively), $\alpha_{C_i} = \min\{2^{\mu_{C_i}}, 2\mu_{C_i}\alpha_v\}$ $(i = 1, \cdots, k)$ and $\alpha_F = \min_i\{2^{\mu_F}, 2\mu_F\alpha_{C_i}\}$. If v is C_i, then $\alpha_F = \min\{2^{\mu_F}, 2\mu_F\alpha_v\}$, $\alpha_M = \min\{2^{\mu_M}, 2\mu_M\alpha_v\}$ and $\alpha_{C_i} = \min\{2^{\mu_{C_i}}, 2\mu_{C_i}\alpha_F, 2\mu_{C_i}\alpha_M\}$ $(i = 1, \cdots, k)$. Thus,

the number of feasible haplotype configuration combinations t_i in trio T_i can be computed consequently, assuming an arbitrary node (node R) as the root of the searching tree. The total number of feasible haplotype configuration combinations in all trios in the pedigree is $t_R = \sum_i t_i$, which can be computed by a tree traversal.

Theorem 3. *Let m_0 be the number of heterozygous loci and t_R be the total number of feasible haplotype configuration combinations for all trios in the pedigree with node R as root. Then the node which gives $\min(t_R)$ can be found in $O(n^2 m_0)$ time and the 1-MRHI problem can be solved in $O(m_0 \min(t_R))$ time.*

Proof. We can evaluate t_R for each node R in the pedigree in $O(nm_0)$ time and choose the node with $\min(t_R)$ as the root in $O(n^2 m_0)$ time. As the computation of numtrio for each feasible haplotype configuration combination in each trio takes $O(m_0)$ time, the 1-MRHI problem can be solved in $O(m_0 \min(t_R))$ time after selecting the best root for the DP algorithm.

4.1 Special Pedigrees with Few Generations

We notice that the diameters of the pedigree graphs in many practical instances are usually small. For example, the 452 families in the CEPH database [1][3][10] consist of only three generations, usually with four grandparents, two parents and a number of children. Figure 4 shows a typical family (family 1413) with 21 members. The longest path starts from one of the grandparents from the father's side to one of the grandparents from the mother's side and is of length 4 (not counting the mating nodes). But if we start from any of the children, we can reach any other node within 2 steps.

Suppose that the number of heterozygous loci in the chosen root R is μ_R, and any other nodes can be reached within l steps from R. We shall enumerate all the $2\mu_R$ feasible haplotype configurations of the root in the first step, and no more than $2^{\mu_R} \times 2m_0$ feasible haplotype configurations for each of its neighboring nodes in the second step, and so on. The number of feasible haplotype configurations

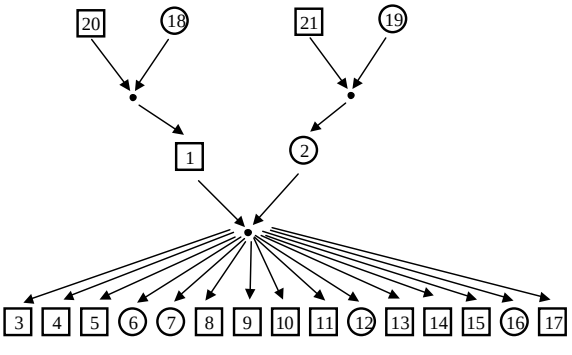


Fig. 4. A typical family(family 1413) in the CEPH database

is at most $2^{\mu_R} \times (2m_0)^l$ at the most distant node. When $\mu_R \ll m_0$ and l is relatively small, we will get an improvement in the time complexity:

Theorem 4. *1-MRHI can be solved in* $\min\{O(nm_0^4 2^{m_0}), O(nm_0^{l+1} 2^{\mu_R+l})\}$ *time for pedigree with n nodes and at most* m_0 *heterozygous loci in each node, where* l *is the maximum path length from the root to the leaves and* μ_R *is the number of heterozygous loci in root* R.

Proof. We have to consider all combinations of feasible haplotype configurations at nodes in each trio, which is at most $2^{\mu_R} \times (2m_0)^l$. We need $O(m_0)$ time to compute *numtrio* for each haplotype configuration combination in each trio, and there may be at most $O(n)$ trios, the time complexity of the algorithm is $\min\{O(nm_0^4 2^{m_0}), O(nm_0^{l+1} 2^{\mu_R+l})\}$.

5 Experimental Results

We implemented the above DP algorithm in C++, and all experiments were conducted on a Pentium IV PC with 1.7GHz CPU and 256MB RAM.

5.1 Real Data

We examined a real data set on Epsiodic Ataxia (EA) by Litt *et al.*[9] which involves a family containing 29 people typed at 9 polymorphic markers on chromosome 12p. Both the locus-based algorithm [4] and the 1-MRHI algorithm run fast ($t < 1$ *sec.*) on this data set but the results are different. The locus-based algorithm gives a feasible solution with 5 recombinations in total but with a double recombination in one haplotype of member 100. The 1-MRHI algorithm, however, finds a more credible solution that has 6 recombinations in total, but with at most 1 recombination for each haplotype in the pedigree.

Another two real data sets are three generations families like those in the CEPH database [1][3][10] (*ftp://genome.wi.mit.edu/distribution/mpg/hapmap/hap_struct/popA/* (*Gabriel et al.*)): family 1331 on chromosome 7a, and family 1346 on chromosome 2a. After removing the loci with missing alleles, family 1331 is a pedigree consisting of 8 members on 32 loci, and family 1346 is a pedigree consisting of 8 members on 55 loci. Both the locus-based algorithm and the 1-MRHI algorithm give the same answer for family 1331, but take 522.4s and 8.7s, respectively. As for family 1346, the locus-based algorithm fails because of not enough resources while the 1-MRHI algorithm finds out a solution in 31 minutes.

5.2 Simulated Data

We compare the performance of our algorithm, with the locus-based algorithm [4] and PHASE [14], a widely used program based on Gibbs Sampling algorithm, the *running time* t and the *accuracy ratio* ρ (in recovering the correct haplotypes). We used three different tree pedigree structures in the experiment: (1) a tree

with 13 nodes (Figure 1), (2) a tree with 29 nodes (Figure 8 in [7]), (3) a typical family with 21 nodes from the CEPH database [1][3][10] (Figure 5).

For each pedigree, genotypes with 15 and 30 biallelic marker loci are considered. The two alleles at each locus of a founder are independently sampled with a fixed frequency of 0.5. The recombination rate is set to $r = 0$, 0.1, 0.2 between generations, and we limit the number of recombinations to no more than one in each PO-pair. For each combination of the above parameters, 100 sets of random genotype data are generated and the average performance of the programs is computed, as shown in Table 1.

Table 1. Comparison of performances of different haplotyping programs on simulation data

	$m = 15$						$m = 30$	
	locus-based[4]		PHASE[14]		1-MRHI		1-MRHI	
(n, r)	$t(sec.)$	ρ	$t(sec.)$	ρ	$t(sec.)$	ρ	$t(sec.)$	ρ
(13, 0.0)	255.7	1.00	688.2	.87	1.68	1.00	202.8	1.00
(29, 0.0)	576.3	1.00	1772.8	.91	12.33	1.00	839.6	1.00
(21, 0.0)	234.4	1.00	592.4	.95	1.02	1.00	44.0	1.00
(13, 0.1)	287.7	.93	972.3	.85	1.73	.91	241.1	.92
(29, 0.1)	542.8	.90	2210.2	.90	10.45	.90	1042.8	.94
(21, 0.1)	243.2	.91	1504.2	.93	0.52	.94	33.7	.96
(13, 0.2)	294.2	.85	1221.4	.85	3.17	.89	1032.4	.86
(29, 0.2)	613.5	.81	3022.2	.89	11.70	.84	916.1	.84
(21, 0.2)	244.1	.90	2106.7	.93	1.22	.95	47.4	.92

† Average performance is obtained from 100 independent executions of each program and for each parameter setting. n stands for the number of nodes, m for the number of marker loci, r for the recombination rate, $t(sec.)$ for the average running time, and ρ for the accuracy ratio.

‡ The locus-based algorithm cannot be applied to cases of $m \geq 30$, due to the space limitation. PHASE is also excluded for cases of $m \geq 30$ because of the time.

As we can see from the table, 1-MRHI runs quickest, and the locus-based algorithm runs quicker than PHASE. Thus the 1-MRHI algorithm can be applied to much larger instances than the locus-based algorithm and PHASE can (the other two algorithms fail when $m = 30$).

In terms of the quality of solutions, all three algorithms can recover the correct haplotype configurations with high probabilities. The accuracy ratio decreases with the increase in the number of recombinations, which is more obvious for the locus-based algorithm and the 1-MRHI algorithm. Since we have limited the number of recombinations within each PO-pair to no more than 1 in the data, the locus-based algorithm, which often finds solutions with fewer recombinations than the actual haplotype configurations, performs worse than the 1-MRHI algorithm as expected.

6 Concluding Remarks

1-MRHI brings an improvement on the running time of solving the general MRHI problem, even though 1-MRHI and the general MRHI usually give the same solutions as confirmed from the experiments. If the solutions are different, 1-MRHI usually gives the more credible solutions. In some cases, if the total number of recombinations for 1-MRHI solutions is much larger than the total number of recombinations for 2-MRHI solutions, then it is plausible that the 2-MRHI solution should be more credible. Our next goal is to find the most probable haplotype configuration which can explain the genotypes in a pedigree when the probabilities of single, double, triple recombinations are given.

Our algorithm for k-MRHI cannot deal with mating loops; nor can the locus-based DP algorithm [4]. A member-based DP algorithm [4] can deal with pedigrees with mating loops, but may not be well-suited to solving the k-MRHI problem because of the increase in time complexity. In practice, pedigree data often contains missing alleles. It will be interesting to find an efficient algorithm for k-MRHI on pedigrees with mating loop and genotypes with missing data.

Acknowledgement. We thank T. Jiang and J. Li for sharing their DP code with us and Dr. M.Y. Chan for proof-reading the first draft of this paper. Thanks to the RGC grant HKU 7135/04E for supporting this research.

References

1. *The CEPH genotype database.* http://www.cephb.fr/.
2. A.G. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Mol. Biol. Evol*, 7(2):111–122, 1990.
3. J. Dausset, H. Cann, D. Cohen, M. Lathrop, J.M. Lalouel, and R. White. Centre d'etude du polymorphisme humain (ceph): collaborative genetic mapping of the human genome. *Genomics*, 5:575–577, 1990.
4. K. Doi, J. Li, and T. Jiang. Minimum recombinant haplotype configuration on tree pedigree. In *Proc. of WABI'03*, pages 339–353, 2003.
5. A. Griffiths, W. Gelbart, R. Lewontin, and J. Miller. *Modern Genetic Analysis: Integrating Genes and Genomes.* W.H. Freeman and Company, N.Y., 2002.
6. D. Gusfield. Inference of haplotypes from samples of diploid populations: complexity and algorithms. *J. Computational Biology*, 8:305–323, 2001.
7. J. Li and T. Jiang. Efficient inference of haplotypes from genotypes on a pedigree. *J. Bioinfo Comp Biol*, 1(1), 2003.
8. Jing Li and Tao Jiang. Efficient inference of haplotypes from genotypes on a pedigree. In *Proc. of RECOMB'03*, pages 197–206, 2003.
9. M. Litt, P. Kramer, D. Browne, S. Gancher, E.R.P. Brunt, D. Root, et al. A gene for episodic ataxia/myokymia maps to chromosome 12p13. *Am. J. Hum. Genet.*, 55:702–709, 1994.
10. J.C. Murray et al. A comprehensive human linkage map with centimorgan density. *Science*, 265:2049–2054, 1994.
11. J.R. O'Connell. Zero-recombinant haplotyping: applications to fine mapping using snps. *Genet Epidemiol*, 19, 2000.

12. D. Qian and L. Beckmann. Minimum-recombinant haplotyping in pedigrees. *Am J Hum Genet*, 70(6):1434–1445, 2002.
13. E. Russo et al. Single nucleotide polymorphism: Big pharma hedges its bets. *The Scientist*, 13, 1999.
14. M. Stephens, N.J. Smith, and P. Donnelly. A new statistical method for haplotype reconstruction for population data. *Am. J. Hum. Genet*, 68:978–989, 2001.
15. P. Tapadar, S. Ghosh, and P.P. Majumder. Haplotyping in pedigrees via a genetic algorithm. *Hum Hered*, 50(1):43–56, 2000.

Calculating Genomic Distances in Parallel Using OpenMP

Vijaya Smitha Kolli[1], Hui Liu[1], Jieyue He[1, 2], Michelle Hong Pan[3], and Yi Pan[1]

[1] Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA
hui.anitaliu@gmail.com, pan@cs.gsu.edu
[2] Department of Computer Science, Southeast University, Nanjing, Jiangsu, China
[3] Centers for Disease Control and Prevention, Office of Workforce and Career Development, Career Development Division, Public Health Informatics Fellow Program, 1600 Clifton Rd, Atlanta, GA 30333, USA
hdp1@cdc.gov

Abstract. By finding the corresponding shortest edit distance between two signed gene permutations, we can know the smallest number of insertions, deletions, and inversions required to change on string of genes into another, where insertion, deletion and inversion are the process of genome evolutions. However, it is NP-hard problem to compute the edit distance between two genomes. Marron et al proposed a polynomial-time approximation algorithm to compute (near) minimum edit distances under inversions, deletions, and unrestricted insertions. Our work is based on Marron's et al algorithm, which carries out lots of comparisons and sorting to calculate the edit distance. These comparisons and sorting are extremely time-consuming, and they result in the decrease of the efficiency. We believe the efficiency of the algorithm can be improved by parallelizing. We parallelize their algorithm via OpenMP on Intel C++ compiler for Linux 7.1, and compare three levels of parallelism: coarse grain, fine grain and combination of both. The experiments are conducted for a varying number of threads and length of the gene sequences. The experimental results have shown that either coarse grain parallelism or fine grain parallelism alone does not improve the performance of the algorithm very much, however, the combination of both fine grain and coarse grain parallelism have improve the performance to a great extent.

1 Introduction

As the need for comparing genomes of different species has grown dramatically with the fast progress of the Human Genome Project, the evolution at the level of whole genomes has attracted more and more attentions from both biologists and computer scientists. They are especially interested in the scenarios in which the genome evolves through insertions, deletions, and movements of genes along its chromosomes.

A gene is the fundamental physical and functional unit of heredity. Each chromosome can be represented by an ordering of signed genes. The gene orders can be rearranged via evolutionary events like inversions and transpositions. The motivation of studying the gene sequencing arises in molecular biology. The ability to

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 113 – 123, 2005.

compare genomes of different species has grown considerably with the rapid advancement of Human Genome Project, genetic and DNA data on different species. One of the most effective methods of finding the similarity between genomes is to compare the order of appearance of identical genes in the two species. By finding the corresponding shortest edit distance between two signed gene permutations, we can know the smallest number of insertions, deletions, and inversions required to change one string of genes into another. However, it is very difficult to compute the edit distance between two genomes; for example, this problem is NP-hard for unsigned permutations even with equal gene content and only inversion allowed.

Many researchers have proposed various algorithms on finding the minimum edit distances [3,4,5,6,10]. Most of these algorithms involve lots of comparisons and sorting while computing edit distances. Marron et al's algorithm [2] is of particular interest for our implementation purposes. This algorithm handles duplications as well as insertions and presents an alternate framework for computing (near) minimal edit sequences involving insertions, deletions, and inversions. They produce a new canonical form in which the shortest edit sequences can be transformed into equivalent sequences of equal length in which all insertions are performed first, following by all inversions, and then by all deletions.

Marron et al's algorithm is implemented sequentially, which is not efficient in terms of computation time, because the algorithm carries out many comparisons and sorting. Parallelism can be employed in the time-consuming comparisons and sorting, thus increasing the efficiency of the algorithm. We have performed profiling on the whole algorithm and identified the functions that are utilizing maximum time. We parallelize the algorithm through OpenMP on Intel® C++ for Linux compiler 7.1. The Intel® Compiler provides optimization technology, threaded application support, features to take advantage of Hyper-Threading technology. At the same time it produces optimal performance for the applications [9]. OpenMP, the industry standard for portable multi-threaded application development, is powerful at fine grain (loop level) and large grain (function level) threading. The Intel C++ Compiler supports OpenMP API version 2.0 and performs code transformation for shared memory parallel programming [9].

The rest of this paper proceeds as follows. Section 2 provides preliminaries on the sequential algorithm by Marron et al. We illustrate the details of our parallel implementation in Section 3. The experimental results are analyzed in Section 4. Section 5 draws the conclusion and proposes future works.

2 Preliminary Existing Algorithm

Marron et al's algorithm [2] is based on a new canonical form for edit sequences. They show that shortest edit sequences can be transformed into equivalent sequences of equal length in which all insertions are performed first, followed by all inversions, and then by all deletions. This canonical form allows taking advantage of El-Mabrouk's exact algorithm for inversions and deletions, which can be extended by finding the best possible prefix of insertions, and producing an approximate solution with bounded error.

2.1 Terminology

The edit sequence is denoted by Greek letter, π.
A string $e_1, e_2, e_3 \ldots e_n$ is contiguous (a clump) iff

$\forall j, e_{j+1} = e_j + 1$.

The parity of a pair of strings (s_i, s_j) is

sign (s_i) . sign (s_j)

The two adjacent substrings with parity in the subject are correctly oriented if they are adjacent with parity in the target.

target = 1 2 3 4 5, subject = -1 2 3 4 -5

substrings (2) and (3 4) are correctly oriented;

substrings (-1) and (2 3 4) are not correctly oriented.

2.2 Canonical Form

Marron et al [2] has proved some positive results about shortest edit sequences. These results will enable to obtain a "*canonical form*" into which any shortest edit sequence can always be transformed without losing optimality. *Reindexing* technique is used in manipulating the order in which operations appear since the order of operations in π need not determine the effect of those operations. Marron et al has proved the following two theorms.

Theorem 1: One two substrings become correctly oriented, they remain correctly oriented.

Theorem 2: All insertions can be done before all inversions and deletions in a Minimum Edit Sequence.

2.3 Sequences Cover

A group of substrings from the target should be determined such that every element in the source appears in one of those substrings. The goal of the job is to cover all the non-deleted target elements with one from the subject. A minimal cover is one that uses fewest substrings of the subject. At each step, we try to cover the target from the left to as far as right as possible with contiguous subsequences of the subject. At last, this method produces a minimal cover by greedy. The cover bound is proved in theorem 3.

Theorem 3: There exists a cover of at most $2|\pi|+1$ for a sequence of S.

2.4 Algorithm Description

El-Mabrouk's approximation method can be applied now by assigning unique labels to all duplicates with the method of constructing a minimal cover. In spite of this, El-Mabrouk's method is used for deletions only, to minimize the error and to make the problem into more easily analyzed form, and later resulting solution is extended to handle the insertions. A new sequence T_{ir} which denotes that all the inserted elements have been removed is obtained by deleting the elements from target sequence T that do not appear in S.

Theorem 4: Let π be the minimal edit sequence from S to T, using l insertions and m inversions. Let π' be the minimal edit sequence of just inversions and deletions from S to T_{ir}. The extension π" (extending π' through an initial sequence of insertions as just described) has at most l +m insertions. Proof. As in [2].

Therefore, if there are *l* insertions and *m* inversions in π, then there are at most $l + m \leq |\pi| = n$ inserted substrings in *T*. Now we can summarize Marron et al 's Genomic Distances algorithm in the following three steps.

Step 1:	Relocate insertions to obtain the canonical form of sequences;
Step 2:	Resolve duplicates by finding the minimum cover through greedy method;
Step 3:	Then run exact EI-Mabrouk algorithm on the inversions and deletions.

3 Details of Parallel Implementation

This section gives the details of the implementation steps carried out to parallelize the Marron et al's genomic distances algorithm.

3.1 Profiling

Profiling is a good procedure to determine the most time-consuming parts of the code. By profiling the sequential code, we obtain a flat profile. Flat profile consists of the percentage of time used to complete the particular function, number of calls made for the function, self milli seconds per call and total milli seconds per call. From this flat profile, we can deduce which function uses greater percentage of time and further which function consumes more total milli seconds per call. The *gprof* utility provides a fast and easy way to do procedural-level profiling of the code. We use *gprof* and focus on the identification of such expensive functions with respect to time, and acknowledge the code to be parallelized. So that, these functions may utilize less time and eventually good *speedup* can be obtained. We compile the code using `-pg` option and run the code as usual. Then the code will produce an output file named gmon.out, which can be analyzed for further purposes.

3.2 Porting the Code Between GNU 3.2 and Intel C++ Compiler 7.1 for Linux

In order to parallelize the algorithm using OpenMP we need to compile the program on Intel C++ compiler because GNU 3.2 does not support OpenMP. For the reason that we are using different compilers, one may expect that each time when the code is ported between GNU 3.2 and Intel C++ 7.1 for Linux, there are problems and bugs, which are previously unknown in the code.

We pursue the following steps in porting the code from GNU 3.2 to Intel C++ Compiler 7.1 for Linux.

- Check all the options used to preprocess, compile, load, and execute the code on both the GNU 3.2 and Intel C++ compiler 7.1 architectures to make sure the same default behavior is being targeted. This involves checking the environment variables and paths.
- Check the level of precision, e.g., how many bits are used in integer arithmetic, and how many bits are used for real and double precision arithmetic.
- Check how string constants are handled.
- Verify whether stack or static storage of variable values is used. If two different methods are being used, this can cause different initialization of variables within routines.
- Check how variables are being initialized (or not initialized) during the building of the code.

There are some significant porting issues encountered while porting program from GNU 3.2 to Intel C++ compiler 7.1. The header files in GNU 3.2 compiler are declared with extension .h as #include < stack.h > but in Intel C++ compiler it is declared without .h extension as #include <stack>.

The initialization of the hash_map variables is much different in GNU compiler and Intel C++ compiler. In GNU, the hash_map is initialized with three parameters, while in Intel C++ compiler it does not accept three parameters in the initialization. Due to this, the code has to be changed to compile successfully on the Intel C++ Compiler without the third parameter.

Following is the sample code for hash_map initializing in Intel C++ compiler 7.1.

```
#include <hash_map>
using namespace std;
void removeDups (int* s1, int &ls1, int* s2, int &ls2)
{
    hash_map< int, int > beenseen1;
    hash_map< int, int> doremap;
    int i; for (i=0; i < ls1; i++)
     {
          if (beenseen1[abs(s1[i])])
               doremap [abs(s1[i])] = 1;
     }
}
```

3.3 Parallel Implementation of Genomic Distances Algorithm

Genomic Distances algorithm is parallelized by using OpenMP on Intel C++ compiler 7.1 for Linux. Parallelism is implemented by applying fine-grain parallelism, coarse-grain parallelism and combination of both. OpenMP method makes use of fork-join technique. Master thread spawns team of threads as needed. Parallelism in OpenMP program can be implemented by adding appropriate directives to change a sequential

program into parallel program. We study how to add appropriate directives to the sequential code to inform OpenMP compiler to parallelize the program.

In coarse-grain parallelism, each function is given attention to find the functional dependencies, which is needed for synchronization, and mutual exclusive sections. Then the independent functions are taken and parallelized using the *parallel* and *section* directive, which specifies that the enclosed section of code are to be divided among the threads in the team and execute them in parallel. The mutually exclusive sections are provided with *critical section* directive, which specifies a region of code that must be executed by only one thread at a time.

In order to implement fine grain parallelism, first data dependency and critical sections within loops is checked to determine whether variables in nested loops should be declared as private or shared in OpenMP program after profiling and porting the program. The variables in the for loops are limited to individual threads, therefore they should be declared as private using *private* directive. Every thread has its own copy of these variables in parallel loops. A variable without the declaration is treated as shared variables by default. Second, the appropriate directive *parallel for* is added in front of loops to inform compilers to execute the code in parallel mode. All threads are scheduled dynamically with varying chunk sizes.

To implement the combination of coarse-grain and fine-grain parallelism in the Genomic Distances algorithm, both the *parallel section* and *parallel for* directive are applied appropriately in the required regions. Special attention is paid to mutual exclusive section.

4 Analysis of Experimental Results

This section discusses the experimental environment and results obtained in implementation of Genomic Distances algorithm via OpenMP. The Algorithm is parallelized by using fine-grain, coarse-grain and combination of both coarse and fine grain parallelism.

4.1 Experimental Environment

An Intel C++ compiler 7.1 is capable of Hyper-Threading technology and supports OpenMP. Thus, we test OpenMP performance via it. This compiler has advanced optimization techniques for the Intel processor. The machine used is the Dell poweredge 6600 server and it has four quad SMP CPU's. Dell server is a powerful scalable parallel system. It is populated with four 1.9 GHz Xeon processors and a total of 4 GB of memory. It has 4 SCSI hard drives under Raid 5. The operating system is Red Hat Linux 8.0 3.2-7. Experiments are conducted by varying the number of threads (2, 4, 6, 8) and length of gene sequences (400, 800, 1000, 2500, 5000, 10000). Speedup and program execution time for OpenMP are measured. Experiment results are verified and made sure that the identical gene sequences are utilized for sequential and parallel program.

4.2 Coarse-Grain Parallelism

As a first step to improve the efficiency of the Genomic Distance algorithm, we adopt OpenMP *parallel sections* construct for all the time-consuming functions. As the *sections*

directive allows performing the sequence of tasks in parallel and assigning each task to a different thread, each function is divided among the available team of threads; the number of threads is specified using *OMP_NUM_THREAD()*. Special attention is paid to the functional dependencies. Fig. 1 compares sequential timing with the coarse-grain parallelism timing with varying the number of threads and length of gene sequences.

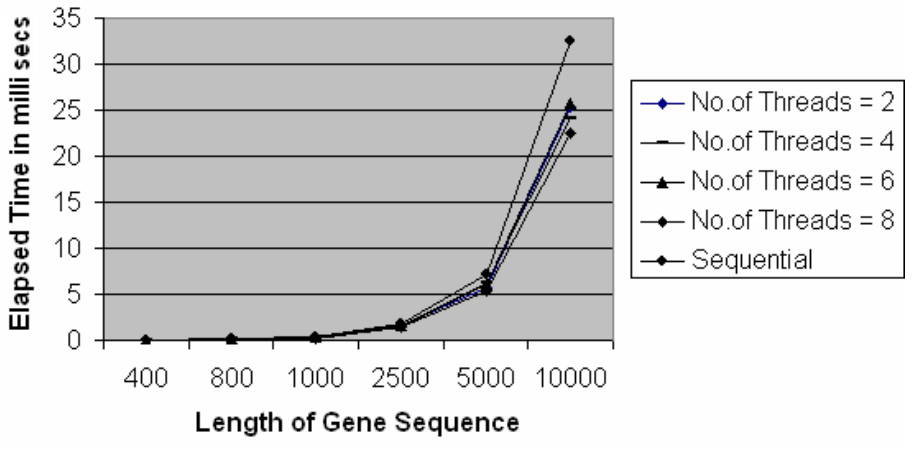


Fig. 1. Average execution time relative to the length of gene sequences under coarse-grain parallelism

Here the sequential algorithm was the most expensive in terms of execution time. We will compare the speedup values among three cases; coarse-grain parallelism, fine-grain parallelism, and the combination of coarse-grain and fine-grain parallelism when the length of gene sequence is 10000, and the number of threads is 8. For coarse-grain parallelism, the speedup value is 1.32. Thus, we conclude by parallelizing the algorithm with the *parallel section* construct there is not much anticipated improvement. Since there is a lot of dependency in the code, as such coarse-grain parallelism by sections does not yield a great performance progress.

4.3 Fine-Grain Parallelism

Later, to improve the efficiency of the Genomic Distance algorithm, we use OpenMP *parallel for* construct for all the time-consuming loops in the program. As the *for* directive specifies that the iterations of the loop immediately following it must be executed in parallel by the team of threads, each for loop will divide the array used into chunks of the specified size dynamically, and the number of threads will work on each individual chunk. With varying sequence size, care is taken to make the chunk size equally distributable among the available threads. Fig. 2 shows the results obtained as a part of fine-grain parallelism.

Here the sequential algorithm is the most consuming in terms of execution time. However, by parallelizing the algorithm with the *parallel for* construct there is a

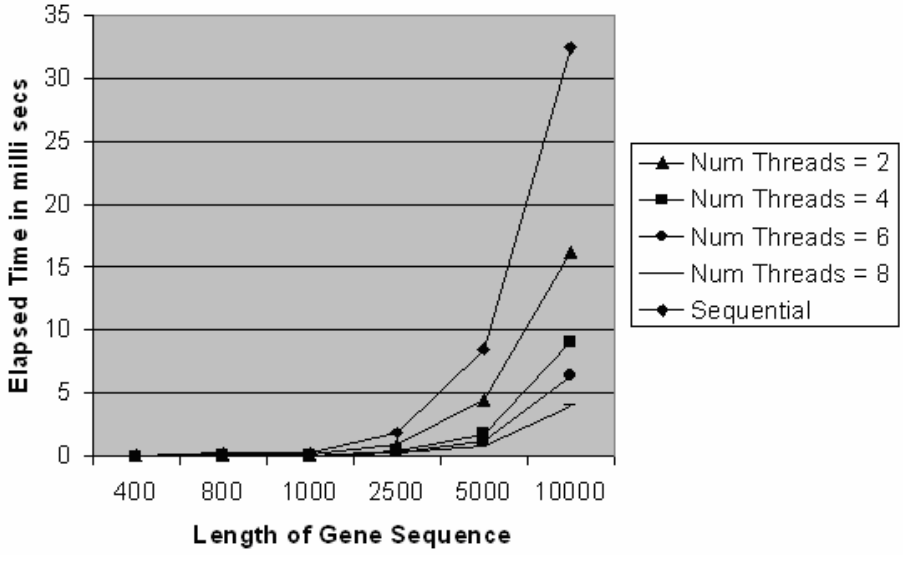


Fig. 2. Average execution time relative to the length of gene sequences under fine-grain parallelism

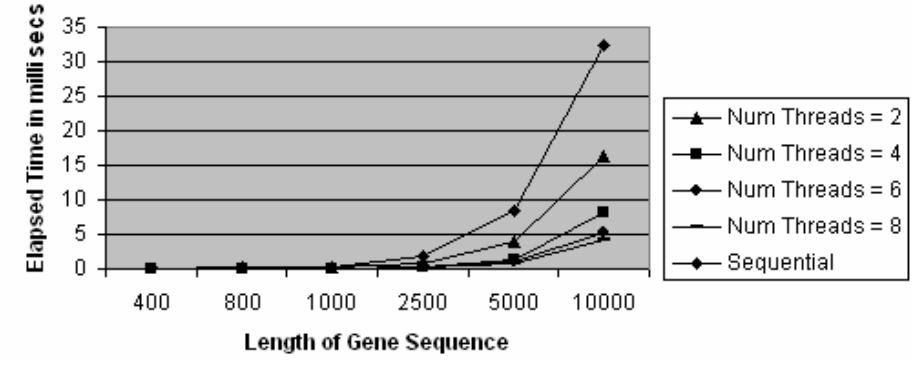


Fig. 3. Average execution time relative to the length of gene sequences under the combination of coarse-grain parallelism and fine-grain parallelism

certain improvement. For example, the speedup value is 5.91 when gene sequence length is 10000, and the number of treads is 8. This is a good improvement compared with only coarse-grain parallelism.

4.4 The Combination of Coarse-Grain Parallelism and Fine-Grain Parallelism

Finally, to get a more efficient Genomic Distance algorithm, we adopt both OpenMP fine-grain and coarse-grain parallelism by implementing both *parallel for* construct and *parallel sections* construct to all the time consuming loops and functions, which

are independent to each other in the program. Fig. 3 shows the results obtained as a part of both fine-grain and coarse-grain parallelism.

As in the earlier cases, the figure shows that the sequential algorithm is the most consuming in terms of execution time. On the other hand, by parallelizing the algorithm with the *parallel for* and *parallel sections* construct there is a very good improvement. For example, the speedup is 7.36 at gene sequence length 10000 and eight threads, which has achieved a much better performance compared with only coarse-grain parallelism and only fine-grain parallelism. Table III gives the speedup values for combination of fine-grain and coarse-grain parallelism for further analysis.

Table 1. Speedup values for combination of coarse-grain and fine-grain parallelism

Len.of Gene Seq.	Number of Threads			
	2	4	6	8
400	3.00	3.00	5.50	6.00
800	2.09	2.27	3.83	5.75
1000	2.07	2.9	4.13	4.83
2500	2.56	5.5	6.48	6.53
5000	2.18	5.07	6.73	7.10
10000	2.00	3.93	6.12	7.36

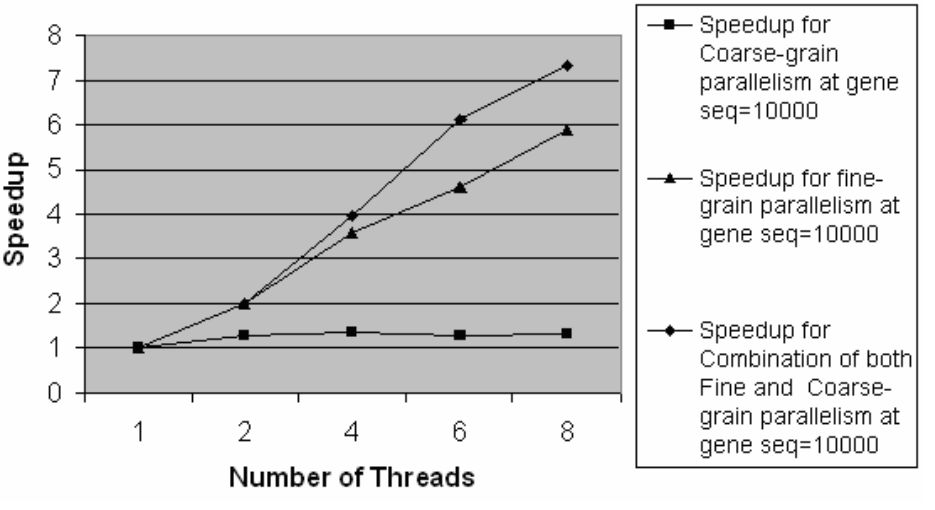


Fig. 4. Comparison of speedup among coarse-grain, fine-grain and combination of both coarse-grain and fine-grain parallelism

The table shows that with smaller gene sequence size the speedups increase slowly with increasing number of threads. But as sequence length increases, there is a precise improvement in the algorithm with maximum speedup in the case of eight threads.

For instance, the speedup value of eight threads is the double of the speedup value of two threads at the length of gene sequence 400. However, the speedup value of eight threads is the triple of that of two threads at the length of gene sequence 10000. At small sequence size, thread creation takes some time, so there is not much of difference in speedup. But as the sequence gets larger, because of equal division of work, speedup is almost as good as (sequential time / number of threads) for a constant sequence size.

Fig. 4 compares the speedup between all three cases when length of gene sequence is 10000. We can observe that the speedup obtained, when the program is implemented with both the fine-grain and coarse-grain parallelism is certainly the best compared to them individually.

By observing all the above three cases, we notice that the combination of both fine-grain and coarse-grain parallelism has improved the performance of the algorithm to a great extent. Because it implements both loop level parallelism, which takes care about all the time-consuming loops, and functional level parallelism, which handles the independent functions and execute them in parallel.

5 Conclusion and Future Work

The Marron et al's Genomic Distances algorithm provides a polynomial-time approximation algorithm with bounded error to compute edit distances under inversions, deletions, and unrestricted insertions from the perfectly sorted sequence to any other. The algorithm consists of many comparisons and sorting, so it is extremely time-consuming. In order to improve the efficiency of the algorithm, we parallelize the algorithm using OpenMP. Furthermore, we study extensively the performance metrics for fine-grain and coarse-grain parallelism and both together.

From our experimental results, we conclude that coarse grain parallelism is not effective for this algorithm since there is lot of functional dependencies, and many functions are not able to execute concurrently. There is improvement in performance when the algorithm is parallelized with fine grain parallelism, for all the time-consuming for loops are made to run in parallel. When the algorithm is parallelized by the combination of both fine and coarse grain parallelism, there is very good improvement in the efficiency of the algorithm.

In the future, we can obtain better performance of the Genomic Distances algorithm by using both MPI and OpenMP. MPI handles the larger-grained communications among multiprocessors, while the lighter-weight threads of OpenMP handle the processor interactions within each multiprocessor. By adding MPI function calls to the OpenMP source program, the program can be transformed into a MPI/OpenMP program suitable for execution on a cluster of multiprocessors.

Acknowledgments

This research was supported in part by the U.S. National Institutes of Health (NIH) under grants R01 GM34766-17S1 and P20 GM065762-01A1, and the U.S. National Science Foundation (NSF) under grants ECS-0196569 and ECS-0334813.

References

1. D.A. Bader, B.M.E. Moret, and M. Yan.: A Fast Linear-time Algorithm for Inversion Distance with an Experimental Comparison. J. Comput. Biol., 8(5):483-491, 2001.
2. Mark Marron, Krister M. Sweson, and Bernard M. E. Moret.: Genomic Distances under Deletions and Insertions. Proc. 9th Int'l Combinatorics and Computing Conf. COCOON'03, 2003, 537-547.
3. A. Caprara.: Sorting by Reversals is Difficult. In Proc. 1st Int'l Conf. on Comput.Mol. Biol. RECOMB97, ACM Press, 1997 75-83.
4. S. Hannenhalli and P. Pevzner.: Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals). In Proc. 27th Ann.Symp. Theory of Computing STOC 95, ACM Press, 1995 178-189.
5. N. El-Mabrouk.: Genome Rearrangement by Reversals and Insertions/Deletions of Contiguous Segments. In Proc. 11th Ann. Symp. Combin. Pattern MatchingCPM 00, volume 1848 of Lecture Notes in Computer Science, Springer-Verlag, 2000 222-234.
6. T. Liu, B.M.E. Moret, and D.A. Bader.: An Exact Linear-time Algorithm for Computing Genomic Distances under Inversions and Deletions *U.* New Mexico, TR-CS-2003-31.
7. http://www.llnl.gov/computing/tutorials/openMP/
8. Michael Quinn, Parallel Programming in C with MPI and OpenMP, The McGraw-Hill Companies, 2004.
9. http://www.intel.com/software/products/compilers/clin/clinux.htm
10. Haim Kaplan, Ron Shamir and Robert E. Tarjan: Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals, Proc. SODA 97 pages 344—351,SIAM Journal on Computing 29 (3) 880--892 (1999).
11. Kai Hwang, Advanced Computer Architecture – Parallelism, Scalability, Programmability, McGraw-Hill, 1993.
12. Tian, X.; Bik,A.; Girkar,M.; Grey,P.; Satio,H.; Su,E.: Intel OpenMP C++/Fortran Compiler for Hyper-Threading Technology: Implementation and Performance. Intel TechnologyJourna. http://developer.intel.com/technology/itj/2002/volume06issue01/ (Feb 2002).
13. T. Dobzhansky and A.H.sturtevant.: Inversions in the Chromosome of Drosophila Pseudoobscura. Genetics, 23:28-64, 1938.
14. D. Bryant.: The Complexity of Calculating Exemplar Distances. In D. Sankoff and J. Nadeau, editors, Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families, pages 207–212. Kluwer Academic Pubs., Dordrecht, Netherlands, 2000.
15. D. Sankoff.: Genome Rearrangement with Gene Families. Bioinformatics, 15(11):909–917, 1999.

Improved Tag Set Design and Multiplexing Algorithms for Universal Arrays*

Ion I. Măndoiu, Claudia Prăjescu, and Dragoş Trincă

CSE Department, University of Connecticut,
371 Fairfield Rd., Unit 2155, Storrs, CT 06269-2155
{ion.mandoiu, claudia.prajescu, dragos.trinca}@uconn.edu

Abstract. In this paper we address two optimization problems arising in the design of genomic assays based on universal tag arrays. First, we address the universal array tag set design problem. For this problem, we extend previous formulations to incorporate antitag-to-antitag hybridization constraints in addition to constraints on antitag-to-tag hybridization specificity, establish a constructive upper bound on the maximum number of tags satisfying the extended constraints, and propose a simple alphabetic tree search tag selection algorithm. Second, we give methods for improving the multiplexing rate in large-scale genomic assays by combining primer selection with tag assignment. Experimental results on simulated data show that this integrated optimization leads to reductions of up to 50% in the number of required arrays.

1 Introduction

High throughput genomic technologies have revolutionized biomedical sciences, and progress in this area continues at an accelerated pace in response to the increasingly varied needs of biomedical research. Among emerging technologies, one of the most promising is the use of *universal tag arrays* [4,7,9], which provide unprecedented assay customization flexibility while maintaining a high degree of multiplexing and low unit cost.

A universal tag array consists of a set of DNA *tags*, designed such that each tag hybridizes strongly to its own *antitag* (Watson-Crick complement), but to no other antitag [2]. Genomic assays based on universal arrays involve multiple hybridization steps. A typical assay [3,5], used for Single Nucleotide Polymorphism (SNP) genotyping, works as follows. (1) A set of *reporter oligonucleotide probes* is synthesized by ligating antitags to the $5'$ end of primers complementing the genomic sequence immediately preceding the SNP location in $3'$-$5'$ order on either the forward or reverse strands. (2) Reporter probes are hybridized in solution with the genomic DNA under study. (3) Hybridization of the primer part ($3'$ end) of a reporter probe is detected by a single-base extension reaction using the polymerase enzyme and dideoxynucleotides fluorescently labeled with 4 different dyes. (4) Reporter probes are separated from the template DNA

* Work supported in part by a "Large Grant" from the University of Connecticut's Research Foundation. A preliminary version of this manuscript appeared in [8].

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 124–137, 2005.

and hybridized to the universal array. (5) Finally, fluorescence levels are used to determine which primers have been extended and learn the identity of the extending dideoxynucleotides.

In this paper we address two optimization problems arising in the design of genomic assays based on the universal tag arrays. First, we address the universal array *tag set design problem* (Section 2). To enable the economies of scale afforded by high-volume production of the arrays, tag sets must be designed to work well for a wide range of assay types and experimental conditions. Ben Dor et al. [2] have previously formalized the problem by imposing constraints on antitag-to-tag hybridization specificity under a hybridization model based on the classical 2-4 rule [10]. We extend the model in [2] to also prevent antitag-to-antitag hybridization and the formation of antitag secondary structures, which can significantly interfere with or disrupt correct assay functionality. Our results on this problem include a constructive upper bound on the maximum number of tags satisfying the extended constraints, as well as a simple alphabetic tree search tag selection algorithm.

Second, we study methods for improving the multiplexing rate (defined as the average number of reactions assayed per array) in large-scale genomic assays involving multiple universal arrays. In general, it is not possible to assign all tags to primers in an array experiment due to, e.g., unwanted primer-to-tag hybridizations. An assay specific optimization that determines the multiplexing rate (and hence the number of required arrays for a large assay) is the *tag assignment problem*, whereby individual (anti)tags are assigned to each primer. In Section 3 we observe that significant improvements in multiplexing rate can be achieved by combining primer selection with tag assignment. For most universal array applications there are multiple primers with the desired functionality; for example in the SNP genotyping assay described above one can choose the primer from either the forward or reverse strands. Since different primers hybridize to different sets of tags, a higher multiplexing rate is achieved by integrating primer selection with tag assignment. This integrated optimization is shown in Section 4 to lead to a reduction of up to 50% in the number of required arrays.

2 Universal Array Tag Set Design

The main objective of universal array tag set design is to maximize the number of tags, which directly determines the number of reactions that can be multiplexed using a single array. Tags are typically required to have a predetermined length [1,7]. Furthermore, for correct assay functionality, tags and their antitags must satisfy the following hybridization constraints:

(H1) Every antitag hybridizes strongly to its tag;
(H2) No antitag hybridizes to a tag other than its complement; and
(H3) There is no antitag-to-antitag hybridization (including hybridization between two copies of the same tag and self-hybridization), since the formation of such duplexes and hair-pin structures prevents corresponding reporter probes from hybridizing to the template DNA and/or leads to undesired primer mis-extensions.

Hybridization affinity between two oligonucleotides is commonly characterized using the *melting temperature*, defined as the temperature at which 50% of the duplexes are in hybridized state. As in previous works [2,3], we adopt a simple hybridization model to formalize constraints (H1)-(H3). This model is based on the observation that stable hybridization requires the formation of an initial *nucleation complex* between two perfectly complementary substrings of the two oligonucleotides. For such complexes, hybridization affinity is well approximated using the classical *2-4 rule* [10], which estimates the melting temperature of the duplex formed by an oligonucleotide with its complement as the sum between twice the number of A+T bases and four times the number of G+C bases.

The *complement* of a string $x = a_1 a_2 \dots a_k$ over the DNA alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$ is $\bar{x} = b_1 b_2 \dots b_k$, where b_i is the Watson-Crick complement of a_{k-i+1}. The *weight* $w(x)$ of x is defined as $w(x) = \sum_{i=1}^{k} w(a_i)$, where $w(\mathtt{A}) = w(\mathtt{T}) = 1$ and $w(\mathtt{C}) = w(\mathtt{G}) = 2$.

Definition 1. *For given constants l, h, and c with $l \leq h \leq 2l$, a set of tags $\mathcal{T} \subseteq \{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}^l$ is called* feasible *if the following conditions are satisfied:*

- *(C1) Every tag in $\mathcal{T}$ has weight h or more.*
- *(C2) Every DNA string of weight c or more appears as substring at most once in the tags of $\mathcal{T}$.*
- *(C3) If a DNA string x of weight c or more appears as a substring of a tag, then $\bar{x}$ does not appear as a substring of a tag unless $x = \bar{x}$.*

The constants l, h, and c depend on factors such as array manufacturing technology and intended hybridization conditions. Property (H1) is implied by (C1) when h is large enough. Similarly, properties (H2) and (H3) are implied by (C1) and (C2) when c is small enough: constraint (C2) ensures that nucleation complexes do not form between antitags and non-complementary tags, while constraint (C3) ensures that nucleation complexes do not form between pairs of antitags.

Universal Array Tag Set Design Problem: *Given constants l, h, and c with $l \leq h \leq 2l$, find a feasible tag set of maximum cardinality.*

Ben-Dor et al. [2] have recently studied a simpler formulation of the problem in which tags of unequal length are allowed and only constraints (C1) and (C2) are enforced. For this simpler formulation, Ben-Dor et al. established a constructive upperbound on the optimal number of tags, and gave a nearly optimal tag selection algorithm based on De Bruijn sequences. Here, we refine the techniques in [2] to establish a constructive upperbound on the number of tags of a feasible set for the extended problem formulation, and propose a simple alphabetic tree search algorithm for constructing feasible tag sets.

The constructive upperbound is based on counting the minimal strings, called *c-tokens*, that can occur as substrings only once in the tags and antitags of a feasible set. Formally, a DNA string x is called c-token if the weight of x is c or more, and every proper suffix of x has weight strictly less than c. The *tail weight* of a c-token is defined as the weight of its last letter. Note that the weight of a c-token can be either c or $c+1$, the latter case being possible only if the c-token starts with a G or a C. As in [2], we use G_n to denote the number of DNA strings

of weight n. It is easy to see that $G_1 = 2$, $G_2 = 6$, and $G_n = 2G_{n-1} + 2G_{n-2}$; for convenience, we also define $G_0 = 1$. In Appendix A we prove the following:

Lemma 1. *Let $c \geq 4$. Then the total number of c-tokens that appear as substrings in a feasible tag set is at most $3G_{c-2} + 6G_{c-3} + G_{\frac{c-3}{2}}$ if c is odd, and at most $3G_{c-2} + 6G_{c-3} + \frac{1}{2}G_{\frac{c}{2}}$ if c is even. Furthermore, the total tail weight of c-tokens that appear as substrings in a feasible tag set is at most*

```
Input: Positive integers c and l, c ≤ l
Output: Feasible MTSDP(l|C|1) solution T
---
Mark all c-tokens as available
For every i ∈ {1, 2, ..., l}, B_i ← A
T ← ∅; Finished ← 0; pos ← 1
While Finished = 0 do
    While the weight of B_1 B_2 ... B_pos < c do
        pos ← pos + 1
    EndWhile
    If the c-token ending B_1 B_2 ... B_pos is available then
        Mark the c-token ending at position pos as unavailable
        If pos = l then
            T ← T ∪ {B_1 B_2 ... B_l}
            pos ← [the position where the first c-token of B_1 B_2 ... B_l ends]
            I ← {i | 1 ≤ i ≤ pos, B_i ≠ G}
            If I = ∅ then
                Finished ← 1
            Else
                pos ← max{I}
                B_i ← A for all i ∈ {pos + 1, ..., l}
                B_pos ← nextbase(B_pos)
            EndIf
        Else
            pos ← pos + 1
        EndIf
    Else
        I ← {i | 1 ≤ i ≤ pos, B_i ≠ G}
        If I = ∅ then
            Mark all the c-tokens in B_1 B_2 ... B_pos−1 as available
            Finished ← 1
        Else
            prevpos ← pos
            pos ← max{I}
            Mark all the c-tokens in B_pos ... B_prevpos−1 as available
            B_i ← A for all i ∈ {pos + 1, ..., l}
            B_pos ← nextbase(B_pos)
        EndIf
    EndIf
EndWhile
```

Fig. 1. The alphabetic tree search algorithm for MTSDP($l|C|1$). The *nextbase*($\cdot$) function is defined by *nextbase*(A) = T, *nextbase*(T) = C, and *nextbase*(C) = G.

$2G_{c-1} + 4G_{c-3} + 2G_{\frac{c-3}{2}}$ *if* c *is odd, and at most* $2G_{c-1} + 4G_{c-3} + G_{\frac{c-2}{2}} + 2G_{\frac{c-4}{2}}$ *if* c *is even.*

Theorem 1. *For every* l, h, c *with* $l \leq h \leq 2l$ *and* $c \geq 4$, *the number of tags in a feasible tag set is at most*

$$\min\left\{\frac{3G_{c-2} + 6G_{c-3} + G_{\frac{c-3}{2}}}{l-c+1}, \frac{2G_{c-1} + 4G_{c-3} + 2G_{\frac{c-3}{2}}}{h-c+1}\right\}$$

for c *odd, and at most*

$$\min\left\{\frac{3G_{c-2} + 6G_{c-3} + \frac{1}{2}G_{\frac{c}{2}}}{l-c+1}, \frac{2G_{c-1} + 4G_{c-3} + G_{\frac{c-2}{2}} + 2G_{\frac{c-4}{2}}}{h-c+1}\right\}$$

for c *even.*

Proof. The proof follows from Lemma 1 by observing that every tag contains at least $l - c + 1$ c-tokens, with a total tail weight of at least $h - c + 1$. □

To generate feasible sets of tags we employ a simple alphabetic tree search algorithm (see Figure 1). A similar algorithm is suggested in [7] for the problem of finding sets of tags that satisfy an unweighted version of constraint (C2). We start with an empty set of tags and an empty tag prefix. In every step we try to extend the current tag prefix t by an additional `A`. If the added letter completes a c-token or a complement of a c-token that has been used in already selected tags or in t itself, we try the next letter in the DNA alphabet, or backtrack to a previous position in the prefix when no more letter choices are left. Whenever we succeed generating a complete tag, we save it and backtrack to the last letter of its first c-token.

3 Improved Multiplexing by Integrated Primer Selection and Tag Assignment

Although constraints (H2)-(H3) in Section 2 prevent unintended antitag-to-tag and antitag-to-antitag hybridizations, the formation of nucleation complexes in-

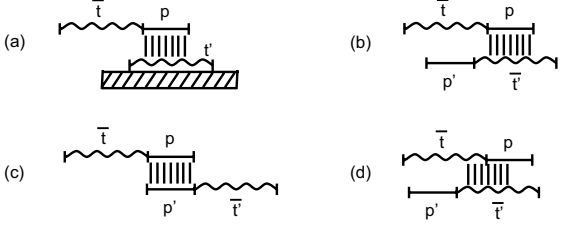


Fig. 2. Four types of undesired hybridizations, caused by the formation of nucleation complexes between (a) a primer and a tag other than the complement of the ligated antitag, (b) a primer and an antitag, (c) two primers, and (d) two reporter probe substrings, at least one of which straddles a ligation point

volving (portions of) the primers may still lead to undesired hybridization between reporter probes and tags on the array (Figure 2(a)), or between two reporter probes (Figure 2(b)-(d)). The formation of these duplexes must be avoided as it leads to extension misreporting, false primer extensions, and/or reduced effective reporter probe concentration available for hybridization to the template DNA or to the tags on the array [3]. This can be done by leaving some of the tags unassigned. As in [3], we focus on preventing primer-to-tag hybridizations (Figure 2(a)). Our algorithms can be easily extended to prevent primer-to-antitag hybridizations (Figure 2(b)); a simple practical solution for preventing the other (less-frequent) unwanted hybridizations is to re-assign offending primers in a post-processing step.

Following [3], a set $\mathcal{P}$ of primers is called *assignable* to a set $\mathcal{T}$ of tags if there is a one-to-one mapping $a : \mathcal{P} \to \mathcal{T}$ such that, for every tag t hybridizing to a primer $p \in \mathcal{P}$, either $t \notin a(\mathcal{P})$ or $t = a(p)$.

Universal Array Multiplexing Problem: *Given primers $\mathcal{P} = \{p_1, \ldots, p_m\}$ and tag set $\mathcal{T} = \{t_1, \ldots, t_n\}$, find a partition of $\mathcal{P}$ into the minimum number of assignable sets.*

For most universal array applications there are multiple primers with the desired functionality, e.g., for the SNP genotyping assay described in Section 1, one can choose the primer from either the forward or reverse strands. Since different primers have different hybridization patterns, a higher multiplexing rate can in general be achieved by integrating primer selection with tag assignment. A similar integration has been recently proposed in [6] between probe selection and physical DNA array design, with the objective of minimizing unintended illumination in photo-lithographic manufacturing of DNA arrays. The idea in [6] is to modify probe selection tools to return *pools* containing all feasible candidates, and let subsequent optimization steps select the candidate to be used from each pool. In this paper we use a similar approach. We say that a set of primer pools is *assignable* if we can select a primer from each pool to form an assignable set of primers.

Pooled Universal Array Multiplexing Problem: *Given primer pools $\mathcal{P} = \{P_1, \ldots, P_m\}$ and tag set $\mathcal{T} = \{t_1, \ldots, t_n\}$, find a partition of $\mathcal{P}$ into the minimum number of assignable sets.*

Let $\mathcal{P}$ be a set of primer pools and $\mathcal{T}$ a tag set. For a primer p (tag t), $\mathcal{T}(p)$ (resp. $\mathcal{P}(t)$) denotes the set of tags (resp. primers of $\bigcup_{P \in \mathcal{P}} P$) hybridizing with p (resp. t). Let $X(\mathcal{P}) = \{P \in \mathcal{P} \ : \ \exists p \in P, \ t \in \mathcal{T}$ s.t. $t \in \mathcal{T}(p)$ and $\mathcal{P}(t) \subseteq P\}$ and $Y(\mathcal{P}) = \{t \in \mathcal{T} \ : \ \mathcal{P}(t) = \emptyset\}$. Clearly, in every pool of $X(\mathcal{P})$ we can find a primer p that hybridizes to a tag t which is not cross-hybridizing to primers in other pools, and therefore assigning t to p will not violate (A1). Furthermore, any primer can be assigned to a tag in $Y(\mathcal{P})$ without violating (A1). Thus, a set $\mathcal{P}$ with $|X(\mathcal{P})| + |Y(\mathcal{P})| \geq |\mathcal{P}|$ is always assignable. The converse is not necessarily true: Figure 3 shows two pools that are assignable although $|X(\mathcal{P})| + |Y(\mathcal{P})| = 0$.

Our primer pool assignment algorithm (see Figure 4) is a generalization to primer pools of Algorithm B in [3]. In each iteration, the algorithm checks whether $|X(\mathcal{P}')| + |Y(\mathcal{P}')| \geq |\mathcal{P}'|$ for the remaining set of pools $\mathcal{P}'$. If not, a primer of maximum *potential* is deleted from the pools. As in [3], the potential

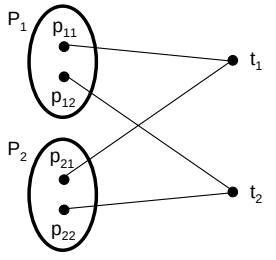


Fig. 3. Two assignable pools for which $|X(\mathcal{P})| + |Y(\mathcal{P})| = 0$

Input: Primer pools $\mathcal{P} = \{P_1, \ldots, P_m\}$ and tag set $\mathcal{T}$
Output: Triples (p_i, t_i, k_i), $1 \leq i \leq m$, where $p_i \in P_i$ is the selected primer for pool i, t_i is the tag assigned to p_i, and k_i is the index of the array on which p_i is assayed

---

$k \leftarrow 0$
While $\mathcal{P} \neq \emptyset$ **do**
  $k \leftarrow k + 1$; $\mathcal{P}' \leftarrow \mathcal{P}$
  While $|X(\mathcal{P}')| + |Y(\mathcal{P}')| < |\mathcal{P}'|$ **do**
    Remove the primer p of maximum potential from the pools in $\mathcal{P}'$
    If p's pool becomes empty **then** remove it from $\mathcal{P}'$
  End While
  Assign pools in $\mathcal{P}'$ to tags on array k
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \mathcal{P}'$
End While

Fig. 4. The iterative primer deletion algorithm

of a tag t with respect to $\mathcal{P}'$ is $2^{-|\mathcal{P}'(t)|}$, and the potential of a primer p is the sum of potentials for the tags in $\mathcal{T}(p)$. If the algorithm deletes the last primer in a pool P, then P is itself deleted from $\mathcal{P}'$; deleted pools are subsequently assigned to new arrays using the same algorithm.

4 Experimental Results

Tag Set Selection. The alphabetic tree search algorithm described in Section 2 can be used to fully or selectively enforce the constraints in Definition 1. In order to assess the effect of various hybridization constraints on tag set size, we ran the algorithm both with constraints (C1)+(C2) and with constraints (C1)+(C2)+(C3). For each set of constraints, we ran the algorithm with c between 8 and 10 for typical practical requirements [1,7] that all tags have length 20 and weight between 28 and 32 (corresponding to a GC-content between 40-60%). We also ran the algorithm with the tag length and weight requirements enforced individually.

Table 1 gives the size of the tag set found by the alphabetic tree search algorithm, as well as the number of c-tokens appearing in selected tags. We also include the theoretical upper-bounds on these two quantities; the upper-bounds for (C1)+(C2) follow from results of [2], while the upper-bounds for

Table 1. Tag sets selected by the alphabetic tree search algorithm

l	$h_{min}/$ h_{max}	c	(C1)+(C2)				(C1)+(C2)+(C3)			
			tags	Bound	c-tokens	Bound	tags	Bound	c-tokens	Bound
20	–/–	8	213	275	2976	3584	107	132	1480	1726
		9	600	816	7931	9792	300	389	3939	4672
		10	1667	2432	20771	26752	844	1161	10411	12780
–	28/32	8	175	224	2918	3584	90	109	1489	1726
		9	531	644	8431	9792	263	312	4158	4672
		10	1428	1854	21707	26752	714	896	10837	12780
20	28/32	8	108	224	1548	3584	51	109	703	1726
		9	333	644	4566	9792	164	312	2185	4672
		10	851	1854	11141	26752	447	896	5698	12780

Table 2. Multiplexing results for $c = 7$ (averages over 10 test cases)

# pools	Pool size	Algorithm	500 tags		1000 tags		2000 tags	
			#arrays	% Util.	#arrays	% Util.	#arrays	% Util.
1000	1	[3]	7.5	30.1	6.0	19.3	5.0	12.1
	2	Primer-Del	6.0	38.7	5.0	24.3	4.1	15.5
	2	Primer-Del+	6.0	39.6	4.5	27.3	4.0	16.5
	2	Min-Pot	6.0	38.4	5.0	24.2	4.0	15.9
	2	Min-Deg	5.8	40.9	4.6	27.0	4.0	16.4
	5	Primer-Del	5.0	49.6	4.0	32.5	3.3	21.0
	5	Primer-Del+	4.0	60.4	3.0	43.6	3.0	24.7
	5	Min-Pot	4.9	50.6	4.0	33.0	3.0	23.5
	5	Min-Deg	4.0	62.0	3.0	44.9	2.7	28.1
2000	1	[3]	13.4	31.8	11.0	19.9	8.7	12.9
	2	Primer-Del	10.7	41.0	8.5	26.4	7.0	16.6
	2	Primer-Del+	10.0	43.3	8.0	28.1	6.0	19.1
	2	Min-Pot	11.0	39.4	9.0	24.8	7.0	16.3
	2	Min-Deg	10.0	43.5	8.0	28.2	6.0	19.2
	5	Primer-Del	8.0	56.8	6.1	38.4	5.0	24.5
	5	Primer-Del+	7.1	62.4	6.0	39.7	4.0	30.1
	5	Min-Pot	9.2	47.5	7.0	32.9	5.0	24.0
	5	Min-Deg	7.0	63.1	5.3	44.2	4.0	30.7
5000	1	[3]	29.5	35.0	23.0	22.6	18.0	14.6
	2	Primer-Del	22.2	47.0	17.1	30.9	13.7	19.6
	2	Primer-Del+	22.2	46.8	17.0	30.9	13.1	20.4
	2	Min-Pot	25.0	41.5	19.2	27.3	15.0	17.7
	2	Min-Deg	22.0	47.3	17.0	31.0	13.0	20.6
	5	Primer-Del	16.6	63.8	12.3	43.9	10.0	27.8
	5	Primer-Del+	16.0	65.6	12.0	44.9	9.0	30.6
	5	Min-Pot	29.5	35.0	23.0	22.6	18.0	14.6
	5	Min-Deg	16.0	65.8	12.0	45.2	9.0	30.8

(C1)+(C2)+(C3) follow from Lemma 1 and Theorem 1. The results show that, for any combination of length and weight requirements, imposing the antitag-to-

antitag hybridization constraints (C3) roughly halves the number of tags selected by the alphabetic tree search algorithm – as well as the theoretical upperbound – compared to only imposing antitag-to-tag hybridization constraints (C1)+(C2). For a fixed set of hybridization constraints, the largest tag sets are found by the alphabetic tree search algorithm when only the length requirement is imposed. The tag weight requirement, which guarantees similar melting temperatures for the tags, results in a 10-20% reduction in the number of tags. However, requiring that the tags have *both* equal length and similar weight results in close to halving the number of tags. This strongly suggests reassessing the need for the strict simultaneous enforcement of the two constraints in current industry designs [1]; our results indicate that allowing small variations in tag length and/or weight results in significant increases in the number of tags.

Integrated Primer Selection and Tag Assignment. We have implemented the iterative primer deletion algorithm in Figure 4 (Primer-Del), a variant of it in which primers in pools of size 1 are omitted – unless all pools have size 1 – when selecting the primer with maximum potential for deletion (Primer-Del+),

Table 3. Multiplexing results for $c = 8$ (averages over 10 test cases)

# pools	Pool size	Algorithm	500 tags #arrays	500 tags % Util.	1000 tags #arrays	1000 tags % Util.	2000 tags #arrays	2000 tags % Util.
1000	1	[3]	3.0	86.0	2.0	77.1	2.0	46.3
	2	Primer-Del	3.0	90.1	2.0	81.6	2.0	47.8
	2	Primer-Del+	3.0	94.5	2.0	88.5	1.0	50.0
	2	Min-Pot	3.0	94.4	2.0	87.9	1.0	50.0
	2	Min-Deg	3.0	92.6	2.0	88.8	1.0	50.0
	5	Primer-Del	3.0	98.0	2.0	92.6	2.0	49.2
	5	Primer-Del+	3.0	99.5	2.0	97.4	1.0	50.0
	5	Min-Pot	3.0	99.4	2.0	97.1	1.0	50.0
	5	Min-Deg	3.0	93.4	2.0	93.4	1.0	50.0
2000	1	[3]	6.0	78.2	4.0	64.4	3.0	48.3
	2	Primer-Del	5.0	92.3	4.0	66.6	3.0	49.8
	2	Primer-Del+	5.0	93.5	3.0	87.9	2.0	78.7
	2	Min-Pot	5.0	93.6	3.0	87.7	2.0	78.1
	2	Min-Deg	5.0	90.8	3.0	87.5	2.0	79.6
	5	Primer-Del	5.0	98.4	3.0	94.1	2.0	84.8
	5	Primer-Del+	5.0	99.5	3.0	97.1	2.0	91.2
	5	Min-Pot	5.0	99.5	3.0	97.0	2.0	90.8
	5	Min-Deg	5.0	91.8	3.0	90.6	2.0	91.7
5000	1	[3]	13.0	81.3	8.6	64.7	6.0	49.3
	2	Primer-Del	12.0	90.5	7.0	81.1	5.0	61.7
	2	Primer-Del+	11.2	93.8	7.0	81.9	4.0	73.8
	2	Min-Pot	12.0	90.4	7.0	81.2	5.0	62.2
	2	Min-Deg	12.0	90.1	7.0	81.5	4.0	73.9
	5	Primer-Del	11.0	98.9	6.0	96.1	4.0	81.7
	5	Primer-Del+	11.0	99.4	6.0	96.8	3.0	97.1
	5	Min-Pot	11.0	99.4	6.0	96.9	4.0	83.1
	5	Min-Deg	11.0	94.6	6.0	91.0	3.4	88.0

and two simple heuristics that first select from each pool the primer of minimum potential (Min-Pot), respectively minimum degree (Min-Deg), and then run the iterative primer deletion algorithm on the resulting pools of size 1. We ran all algorithms on data sets with between 1000 to 5000 pools of up to 5 randomly generated primers. As in [3], we varied the number of tags between 500 and 2000.

For instance size, we report the number of arrays and the average tag utilization (computed over all arrays except the last) obtained by (a) algorithm B in [3] run using a single primer per pool, (b) the four pool-aware assignment algorithms run with 1 additional candidate in each pool, and (c) the four pool-aware assignment algorithms run with 4 additional candidates in each pool. Scenario (b) models SNP genotyping applications in which the primer can be selected from both strands of the template DNA, while scenario (c) models applications such as gene transcription monitoring, where significantly more than 2 gene specific primers are typically available.

In a first set of experiments we extracted tag sequences from the tag set of the commercially available GenFlex Tag Arrays. All GenFlex tags have length 20; primers used in our experiments are 20 bases long as well. Primer-to-tag hybridizations were assumed to occur between primers and tags containing complementary c-tokens with $c = 7$ (Table 2), respectively $c = 8$ (Table 3). The results show that significant improvements in multiplexing rate – and a corresponding reduction in the number of arrays – are achieved by the pool-aware algorithms over the algorithm in [3]. For example, assaying 5000 reactions on a 2000-tag array requires 18 arrays using the method in [3] for $c = 7$, compared to only 13 (respectively 9) if 2 (respectively 5) primers per pool are available.

Table 4. Multiplexing results (averages over 10 test cases) for two sets of 213 tags of length 20, one constructed by running the alphabetic tree search algorithm in Section 2 with $c = 8$ and constraints (C1)+(C2), and the other extracted from the GenFlex Tag Array

# pools	Pool size	Algorithm	GenFlex tags #arrays	GenFlex tags % Util.	Tree search tags #arrays	Tree search tags % Util.
1000	1	[3]	6.0	90.0	5.0	100.0
	2	Primer-Del+	5.0	100.0	5.0	100.0
	2	Min-Deg	5.9	94.0	5.0	100.0
	5	Primer-Del+	5.0	100.0	5.0	100.0
	5	Min-Deg	5.2	97.3	5.0	100.0
2000	1	[3]	11.0	90.6	10.0	99.2
	2	Primer-Del+	10.0	98.7	10.0	100.0
	2	Min-Deg	10.8	94.2	10.0	99.3
	5	Primer-Del+	10.0	100.0	10.0	100.0
	5	Min-Deg	10.1	96.0	10.0	99.3
5000	1	[3]	26.5	91.3	24.0	99.2
	2	Primer-Del+	25.0	97.6	24.0	100.0
	2	Min-Deg	25.0	96.3	24.0	99.3
	5	Primer-Del+	24.0	100.0	24.0	100.0
	5	Min-Deg	25.0	96.6	24.0	99.3

In these experiments, the Primer-Del+ algorithm dominates in solution quality the Primer-Del, while Min-Deg dominates Min-Pot. Neither Primer-Del+ nor Min-Deg consistently outperforms the other over the whole range of parameters, which suggests that a good practical meta-heuristic is to run both of them and pick the best solution obtained.

In a second set of experiments we compared two sets of 213 tags of length 20, one constructed by running the alphabetic tree search algorithm in Section 2 with $c = 8$ and constraints (C1)+(C2), and the other extracted from the GenFlex Tag Array. The results in Table 4 show that the tags selected by the alphabetic tree search algorithm participate in fewer primer-to-tag hybridizations, which leads to an improved multiplexing rate.

References

1. Affymetrix, Inc. GeneFlex tag array technical note no. 1, available online at http://www.affymetrix.com/support/technical/technotes/genflex_technote.pdf.
2. A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini. Universal DNA tag systems: a combinatorial design scheme. *Journal of Computational Biology*, 7(3-4):503–519, 2000.
3. A. BenDor, T. Hartman, B. Schwikowski, R. Sharan, and Z. Yakhini. Towards optimally multiplexed applications of universal DNA tag systems. In *Proc. 7th Annual International Conference on Research in Computational Molecular Biology*, pages 48–56, 2003.
4. S. Brenner. Methods for sorting polynucleotides using oligonucleotide tags. *US Patent 5,604,097*, 1997.
5. J.N. Hirschhorn et al. SBE-TAGS: An array-based method for efficient single-nucleotide polymorphism genotyping. *PNAS*, 97(22):12164–12169, 2000.
6. A.B. Kahng, I.I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky. Design flow enhancements for DNA arrays. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 116–123, 2003.
7. M.S. Morris, D.D. Shoemaker, R.W. Davis, and M.P. Mittmann. Selecting tag nucleic acids. *U.S. Patent 6,458,530 B1*, 2002.
8. I.I. Măndoiu, C. Prăjescu, and D. Trincă. Improved tag set design and multiplexing algorithms for universal arrays. In V.S. Sunderam et al., editor, *Proc. IWBRA 2005/ICCS 2005*, volume 3515 of *Lecture Notes in Computer Science*, pages 994–1002, Berlin, 2005. Springer-Verlag.
9. N.P. Gerry et al. Universal DNA microarray method for multiplex detection of low abundance point mutations. *J. Mol. Biol.*, 292(2):251–262, 1999.
10. R.B. Wallace, J. Shaffer, R.F. Murphy, J. Bonner, T. Hirose, and K. Itakura. Hybridization of synthetic oligodeoxyribonucleotides to phi chi 174 DNA: the effect of single base pair mismatch. *Nucleic Acids Res.*, 6(11):6353–6357, 1979.

A Proof of Lemma 1

We first establish two lemmas on self-complementary DNA strings, i.e., strings $x \in \{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}^+$ with $x = \overline{x}$.

Lemma 2. *If x is self-complementary then $|x|$ and $w(x)$ are both even.*

Proof. Let $x = x_1 x_2 \ldots x_p$ be a self-complementary DNA string. If $p = 2q+1$, by the definition of the complement we should have $x_{q+1} = \overline{x}_{q+1}$, which is impossible. Thus, $p = 2q$. Since $x_1 = \overline{x}_{2q}$,$x_2 = \overline{x}_{2q-1}$,..., $x_q = \overline{x}_{q+1}$, and the weight of complementary bases is the same, it follows that $w(x) = 2\sum_{i=1}^{q} w(x_i)$. □

Lemma 3. *Let H_n be the number of self-complementary DNA strings of weight n. $H_n = 0$ if n is odd, and $H_n = G_{n/2}$ if n is even.*

Proof. By Lemma 2, self-complementary strings must have even length and weight. For even n, the mapping $x_1 \ldots x_q x_{q+1} \ldots x_{2q} \mapsto x_1 \ldots x_q$ gives a one-to-one correspondence between self-complementary strings of weight n and strings of weight $n/2$. □

Proof of Lemma 1. Let W and S denote weak and strong DNA bases (A or T, respectively G or C), and let $<w>$ denote the set of DNA strings with weight w. The c-tokens can be partitioned into the seven classes given in Table 5, depending on total token weight (c or $c+1$) and the type of starting and ending bases. This partitioning is defined so that, for every c-token x, the class of the unique c-token suffix of $\overline{x}$ can be determined from the class of x. Note that $\bar{x}$ is itself a c-token, except when $x \in \mathtt{S}{<}c-3{>}\mathtt{WW} \cup \mathtt{S}{<}c-4{>}\mathtt{SW}$.

Table 5. Classes of c-tokens

Class of x	c-token suffix of $\overline{x}$
W<$c-3$>S	S<$c-3$>W
S<$c-4$>S	S<$c-4$>S
S<$c-3$>S	S<$c-3$>S
W<$c-2$>W	W<$c-2$>W
S<$c-3$>W	W<$c-3$>S
S<$c-3$>WW	W<$c-3$>S
S<$c-4$>SW	S<$c-4$>S

Let N_{cls} denote the number of c-tokens of class *cls* occurring in a feasible tag set.

A.1 c Odd

Since W<$c-3$>S ∪ S<$c-3$>W can be partitioned into $4G_{c-3}$ pairs $\{x, \bar{x}\}$ of complementary c-tokens, and at most one token from each pair can appear in a feasible tag set,

$$N_{\mathtt{W}<c-3>\mathtt{S}} + N_{\mathtt{S}<c-3>\mathtt{W}} \leq 4G_{c-3} \tag{1}$$

Similarly, class W<$c-2$>W can be partitioned into $2G_{c-2}$ pairs $\{x, \bar{x}\}$ of complementary c-tokens, W<$c-3$>S ∪ S<$c-3$>WW can be partitioned into $4G_{c-3}$ triples $\{x, \bar{x}A, \bar{x}T\}$ with $x \in \mathtt{W}{<}c-3{>}\mathtt{S}$, S<$c-3$>W ∪ S<$c-3$>WW can be partitioned into $4G_{c-3}$ triples $\{x, xA, xT\}$ with $x \in \mathtt{S}{<}c-3{>}\mathtt{W}$, and S<$c-4$>S ∪

$\mathtt{S}\!<\!c-4\!>\!\mathtt{SW}$ can be partitioned into $2G_{c-4}$ 6-tuples $\{x, \bar{x}, xA, xT, \bar{x}A, \bar{x}T\}$ with $x \in \mathtt{S}\!<\!c-4\!>\!\mathtt{S}$. Since at most one c-token can appear in a feasible tag set from each such pair, triple, respectively 6-tuple,

$$N_{\mathtt{W}<c-2>\mathtt{W}} \leq 2G_{c-2} \tag{2}$$

$$N_{\mathtt{W}<c-3>\mathtt{S}} + N_{\mathtt{S}<c-3>\mathtt{WW}} \leq 4G_{c-3} \tag{3}$$

$$N_{\mathtt{S}<c-3>\mathtt{W}} + N_{\mathtt{S}<c-3>\mathtt{WW}} \leq 4G_{c-3} \tag{4}$$

$$N_{\mathtt{S}<c-4>\mathtt{S}} + N_{\mathtt{S}<c-4>\mathtt{SW}} \leq 2G_{c-4} \tag{5}$$

Using Lemma 3, it follows that $\mathtt{S}\!<\!c-3\!>\!\mathtt{S}$ contains $2G_{\frac{c-3}{2}}$ self-complementary c-tokens. Since the remaining $4G_{c-3} - 2G_{\frac{c-3}{2}}$ c-tokens can be partitioned into complementary pairs each contributing at most one c-token to a feasible tag set,

$$N_{\mathtt{S}<c-3>\mathtt{S}} \leq \frac{1}{2}\left(4G_{c-3} - 2G_{\frac{c-3}{2}}\right) + 2G_{\frac{c-3}{2}} = 2G_{c-3} + G_{\frac{c-3}{2}} \tag{6}$$

Adding inequalities (1), (3), and (4) multiplied by 1/2 with (2), (5), and (6) implies that the total number of c-tokens in a feasible tag set is at most

$$2G_{c-2} + 8G_{c-3} + 2G_{c-4} + G_{\frac{c-3}{2}} = 3G_{c-2} + 6G_{c-3} + G_{\frac{c-3}{2}}$$

Furthermore, adding (1), (2), and (3) with inequalities (5) and (6) multiplied by 2 implies that the total tail weight of the c-tokens in a feasible tag set is at most

$$2G_{c-2} + 12G_{c-3} + 4G_{c-4} + 2G_{\frac{c-3}{2}} = 2G_{c-1} + 4G_{c-3} + 2G_{\frac{c-3}{2}}$$

A.2 c Even

Inequalities (1), (3), and (4) continue to hold for even values of c. Since $c-3$ is odd, $\mathtt{S}\!<\!c-3\!>\!\mathtt{S}$ contains no self-complementary tokens and can be partitioned into $2G_{c-3}$ pairs $\{x, \bar{x}\}$,

$$N_{\mathtt{S}<c-3>\mathtt{S}} \leq 2G_{c-3} \tag{7}$$

By Lemma 3, there are $2G_{\frac{c-4}{2}}$ self-complementary tokens in $\mathtt{S}\!<\!c-4\!>\!\mathtt{S}$. Therefore $\mathtt{S}\!<\!c-4\!>\!\mathtt{S} \cup \mathtt{S}\!<\!c-4\!>\!\mathtt{SW}$ can be partitioned into $2G_{\frac{c-4}{2}}$ triples $\{x, xA, xT\}$ with $x \in \mathtt{S}\!<\!c-4\!>\!\mathtt{S}$, $x = \bar{x}$ and $2G_{c-4} - G_{\frac{c-4}{2}}$ 6-tuples $\{x, \bar{x}, xA, xT, \bar{x}A, \bar{x}T\}$ with $x \in \mathtt{S}\!<\!c-4\!>\!\mathtt{S}$, $x \neq \bar{x}$. Since a feasible tag set can use at most one c-token from each triple and 6-tuple,

$$N_{\mathtt{S}<c-4>\mathtt{S}} + N_{\mathtt{S}<c-4>\mathtt{SW}} \leq 2G_{c-4} + G_{\frac{c-4}{2}} \tag{8}$$

Using again Lemma 3, we get

$$N_{\mathtt{W}<c-2>\mathtt{W}} \leq 2G_{c-2} + G_{\frac{c-2}{2}} \tag{9}$$

Adding inequalities (1), (3), and (4) multiplied by 1/2 with (7), (8), and (9) implies that the total number of c-tokens in a feasible tag set is at most

$$2G_{c-2} + 8G_{c-3} + 2G_{c-4} + G_{\frac{c-2}{2}} + G_{\frac{c-4}{2}} = 3G_{c-2} + 6G_{c-3} + \frac{1}{2}G_{\frac{c}{2}}$$

Finally, adding (1), (3), and (9) with inequalities (7) and (8) multiplied by 2 implies that the total tail weight of the c-tokens in a feasible tag set is at most

$$2G_{c-2} + 12G_{c-3} + 4G_{c-4} + G_{\frac{c-2}{2}} + 2G_{\frac{c-4}{2}} = 2G_{c-1} + 4G_{c-3} + G_{\frac{c-2}{2}} + 2G_{\frac{c-4}{2}}$$

□

Virtual Gene: Using Correlations Between Genes to Select Informative Genes on Microarray Datasets*

Xian Xu and Aidong Zhang

Department of Computer Science and Engineering,
State University of New York at Buffalo, Buffalo, NY 14260, USA
{xianxu, azhang}@cse.buffalo.edu

Abstract. Gene Selection is one class of most used data analysis algorithms on microarray datasets. The goal of gene selection algorithms is to filter out a small set of informative genes that best explains experimental variations. Traditional gene selection algorithms are mostly single-gene based. Some discriminative scores are calculated and sorted for each gene. Top ranked genes are then selected as informative genes for further study. Such algorithms ignore completely correlations between genes, although such correlations is widely known. Genes interact with each other through various pathways and regulative networks. In this paper, we propose to use, instead of ignoring, such correlations for gene selection. Experiments performed on three public available datasets show promising results.

1 Introduction

Microarray experiments enable biologists to monitor expression levels of thousands of genes or ESTs simultaneously [1,7,18]. Short sequences of genes or ESTs tagged with fluorescent materials are printed on a glass surface. The slice is then exposed to sample solution for hybridization (base-pairing). mRNA molecules are expected to hybridize with short sequences matching part of their complement sequences. After hybridization the slice is scanned and goes through various data processing steps including image processing, quality control and normalization [4]. The resulting dataset is a two dimensional array with thousands of rows (genes) and tens of columns (experiments). Element at i^{th} row and j^{th} column in such an array is the expression level measure for gene i in experiment j. When tissue samples used in the experiments are labeled (e.g., sample is cancer tissue or normal tissue), sample classification can be performed on such dataset. New samples are classified based on their gene expression profiles.

Such dataset poses special challenge for pattern recognition algorithms. The main obstacle is the limited number of samples due to practical and financial concerns. This results in the situation where the number of features (or genes) well outnumbers

* This research is partly supported by National Science Foundation Grants DBI-0234895, IIS-0308001 and National Institutes of Health Grant 1 P20 GM067650-01A1. All opinions, findings, conclusions and recommendations in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation or the National Institutes of Health.

C. Priami, A. Zelikovsky (Eds.): Trans. on Comput. Syst. Biol. II, LNBI 3680, pp. 138–152, 2005.

the number of observations. The term "curse of dimensionality" and "peaking phenomenon" are coined in the machine learning and pattern recognition community, referring to the phenomenon that inclusion of excessive features may actually degrade the performance of a classifier if the number of training examples used to build the classifier is relatively small compared to the number of features [11]. Typical treatment is to reduce the dimensionality of the feature space before classification using feature extraction and feature selection. Feature extraction algorithms create new features based on transformation and/or combination of original features while feature selection algorithms aim to select a subset of original features. Techniques like PCA and SVD have been used to create salient features [9,13] for sample classification on microarray datasets. Feature selection, or in our case, gene selection generates a small set of informative genes, which not only leads to better classifiers, but also enables further biological investigation [8,14,16].

In order to find the optimal subset of features that maximizes some feature selection criterion function (we assume the higher value the criterion function, the better the feature subset), straightforward implementation would require evaluation of the criterion function for each feature subset, which is a classic NP hard problem. Various heuristics and greedy algorithms have been proposed to find sub-optimal solutions. Assuming independence between features, one attempt is to combine small feature subsets with high individual scores. This heuristic is widely used for gene selection. A class of gene selection algorithms calculates discriminative scores for individual genes and combines top ranked genes as selected gene set. We refer to this class of algorithms single gene based algorithms. Various discriminative scores have been proposed, including statistical tests (t-test, F-test) [3], non-parametric tests like TNoM [2], mutual information [22,23], S2N ratio (signal to noise ratio) [7] , extreme value distribution [15] and SAM [19] etc. Although simple, this class of algorithms is widely used in microarray data analysis and proven to be effective and efficient.

However, the assumption of independence between genes over simplifies the complex relationship between genes. Genes are well known to interact with each other through gene regulative networks. As a matter of fact, the common assumption of cluster analysis on microarray dataset [12] is that co-regulated genes have similar expression profiles. Bø [3] proposed to calculate discriminant scores for a pair of genes instead of each individual gene. Several of recent researches on feature selection especially gene selection [10,20,21,23] took into consideration the correlation between genes explicitly by limiting redundancy in resulting gene set. Heuristically, selected genes need to first have high discriminative scores individually and secondly not correlate much with genes that have already been selected. Generic feature selection algorithms like SFFS (sequential forward floating selection), SBFS (sequential backward floating selection), etc. have also been used for selecting informative genes from microarray datasets.

In this paper, we propose a totally different approach. Instead of trying to get rid of correlation in the selected gene set, we examine whether such correlation itself is a good predictor of sample class labels. Our algorithm is a supervised feature extraction algorithm based on new feature "virtual gene". "Virtual genes" are linear combinations of real genes on a microarray dataset. Top ranked "virtual genes" are used for further analysis, e.g., sample classification. Our experiments with three public available datasets

suggest that correlations between genes are indeed very good predictors of sample class labels. Unlike typical feature extraction algorithms, the "virtual gene" bears biological meaning: the weighted summation or difference of expression levels of several genes.

The rest of this paper is organized as follows. We present the concept of "virtual gene" and the "pairwise virtual gene" algorithm in Sec. 2. Both a synthetic and a real example from Alon dataset [1] are given. In Sec. 3, extensive experimental results are reported using three public available datasets. We give our conclusion and future work of this paper in Sec. 4.

2 Virtual Gene: A Gene Selection Algorithm

2.1 Gene Selection for Microarray Experiments

In this section we formalize the problem of gene selection for microarray datasets. Symbols used in this section will be used throughout this paper.

Let $\mathcal{G}$ be the set of all genes that are used in one study, $\mathcal{S}$ be the set of all experiments performed, $\mathcal{L}$ be the set of sample class labels of interest. We assume $\mathcal{G}, \mathcal{S}, \mathcal{L}$ are fixed for any given study. Let $n = |\mathcal{G}|$ be the total number of genes, $m = |\mathcal{S}|$ be the total number of experiments and $l = |\mathcal{L}|$ be the total number of class labels. A gene expression dataset used in our study can be defined as $\mathcal{E} = (\mathcal{G}, \mathcal{S}, L, E)$, where L is a list of sample class labels such that for $s \in \mathcal{S}$, $L(s) \in \mathcal{L}$ is the class label for sample s; expression matrix E is an $n \times m$ matrix of real numbers. $E(g,s)$, where $g \in \mathcal{G}, s \in \mathcal{S}$, is the expression level of gene g in experiment s. For simplicity of presentation, we use a subscripting scheme to refer to elements in $\mathcal{E}$. Let $\mathcal{E}(G,S) = (G,S,L',E')$ where $G \subseteq \mathcal{G}$ and $S \subseteq \mathcal{S}$. L' is a sublist of L containing class labels for samples S, E' is the subarray of E containing values of expression levels for genes G and experiments S. We also write $E' = E(G,S)$. We further use $L(S)$ to denote a list of class labels for the set of experiments S. Given training expression data $\mathcal{E}_{train} = (\mathcal{G}, S_{train}, L_{train}, E_{train})$, the problem of sample classification is to build a classifier that predicts L_{new} for new experiment result $\mathcal{E}_{new} = (\mathcal{G}, S_{new}, L_{missing}, E_{new})$. $L_{missing}$ indicates that the class labels of samples S_{new} have not been decided yet. The problem of gene selection is to select a subset of genes $G' \subset \mathcal{G}$ based on $\mathcal{E}_{train}$ so that classifiers built from $\mathcal{E}_{train}(G', S_{train})$ predict L_{new} more accurately than classifiers built from $\mathcal{E}_{train}$. We use n' as the number of features being selected, or $n' = |G'|$.

2.2 An Example

Consider the following two examples as shown in Figure 1. In each figure, the expression levels of two genes are monitored across several samples. Samples are labeled either cancerous or normal. In both cases, the expression levels of the selected genes vary randomly across the sample classes. However, their correlation is a good predictor of class labels. *Virtual gene* expression level is obtained using the Def. 2. In the case of Alon [1] dataset, the expression levels of H09719 are generally higher than that of L07648 in cancer tissues. In normal tissues, on the contrary, L07648 expresses consistently higher except in one sample. Such correlations could be good predictors of

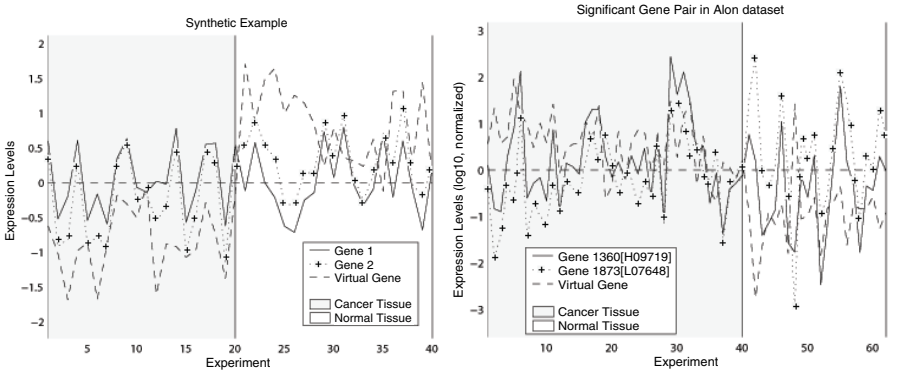


Fig. 1. Examples of gene pair being better predictor of class labels than single gene

sample class labels. However, all feature selection algorithms listed in the previous section can not find and use such correlations. Single gene based algorithms will ignore both genes since neither of them is a good predictor of sample class labels in its own right. Correlation based algorithms will actually remove such correlations, should any of the genes have been selected.

2.3 Virtual Gene Algorithm

Definition 1. ***Virtual Gene*** *is a triplet* $VG = (G_v, W, b)$ *where* $G_v \subseteq \mathcal{G}$ *is a set of constituent genes,* $|G_v| = n_v$*,* W *is a matrix of size* $n_v \times 1$*,* b *is a numeric value. The expression levels of a* ***virtual gene*** *is determined using Definition 2.*

Definition 2. ***(Virtual Gene Expression)*** *Given a* ***virtual gene*** $VG = (G_v, W, b)$ *and gene expression matrix* E*, where* $|G_v| = n_v$*,* E *is an* $n_v \times m_v$ *expression matrix, the* ***virtual gene expression*** VE *of a* virtual gene VG *is a linear combination of expression matrix* E*.* $VE(VG, E) = W' \times E + b$*, where* W' *is the transpose of* W*.*

A *virtual gene* is a triplet $VG = (G, W, b)$ as defined in Def. 1. Parameters W and b are chosen using FLD (fisher linear discriminant) to maximize linear separability between sample classes as listed in Algorithm 1. Discriminative power of a *virtual gene expression* with respect to sample classes can be measured using normal single gene based scores. We use t-score in this paper for this purpose. *Pairwise virtual gene* is a special case of *virtual gene* where the number of genes involved is limited to two. In this case, only the correlations between a pair of genes are considered. By limiting

Algorithm 1 *gen_vg* : Calculating Virtual Gene From Training Data

Require: $\mathcal{E} = (G, S, L, E)$ as gene expression data.
Ensure: $VG = (G, W, b)$ as a virtual gene.
1: $(W, b) \leftarrow fld(E, L)$, (W, b) is the model returned by fld algorithm.
2: **return** (G, W, b)

virtual gene to gene pairs, computation can be carried out efficiently. According to our experiments, it performs well on three public available datasets.

Definition 3. ***Pairwise virtual gene and its expression*** *are special cases for* ***virtual gene*** *and its expression, where the number of genes involved is limited to two.*

Exhaustive examination of all *pairwise virtual genes* requires $O(n^2)$ computation where n is the number of genes. For a large number of genes, exhaustive search of all gene pairs becomes inefficient. Such exhaustive search also invites unwanted noise since not all gene pairs bare biological meaning. For example, for genes that are expressed in different locations in a cell, in different biological processes, without biological interactions, their relative abundance may not be biologically significant. Ideally, only gene pairs with some biological interaction shall be examined. We approximate this using a gene clustering approach. Each gene cluster corresponds roughly to some biological pathways. By limiting search among the gene pairs from the same gene cluster, we not only focus ourselves on these gene pairs that are more likely to interact biologically, but also make our gene selection algorithm much faster.

Algorithm 2 details the *pairwise virtual gene selection* algorithm. Genes are first clustered based on their expression levels. For each pair of genes in the same cluster, virtual gene expression is calculated according to Def. 2. A single gene discriminative score with respect to the sample class labels is then derived from the virtual gene expression. All within-cluster pairwise virtual gene expression scores are calculated and stored

Algorithm 2 *pairwise_vg* : Pairwise Virtual Gene Selection

Require: $\mathcal{E} = (G, S, L, E)$; k as the number of genes to be selected; α; β
Ensure: VGS: as set of pairwise virtual genes $VG = (G, W, b)$

Initialize VGS to be an empty set. Initialize *pair_score* to be a sparse $n \times n$ array.
Cluster genes based on their expression levels in E. Result stores in *Clusters*.
for each gene cluster $G' \in$ *Clusters* **do**
  for all gene $g1 \in G'$ **do**
    for all gene $g2 \in G'$ and $g2 \neq g1$ **do**
      $vg \leftarrow gen\_vg(\mathcal{E}((g1, g2), S))$
      $ve \leftarrow \mathrm{VE}(vg, E((g1, g2), S))$
      $pair\_score[g1, g2] \leftarrow t\text{-score}(ve, L)$
    end for
  end for
end for
for $i = 1$ to k **do**
  $(g1, g2) \leftarrow \underset{(g1,g2)}{\mathrm{argmax}}(pair\_score[g1, g2])$
  $vg \leftarrow gen\_vg(\mathcal{E}((g1, g2), S))$
  add vg to VGS
  multiply *pair_score* that involves $g1$ or $g2$ by α.
  multiply *pair_score* that involves genes in same cluster of $g1$ or $g2$ by β.
  $pair\_score[g1, g2] \leftarrow$ minimum value
end for
return VGS

for the next stage of analysis. The best scored virtual gene is then selected and pairwise scores are modified by two parameters. Pairwise scores of virtual genes that share constituent genes with the selected virtual gene are degraded by a constant α ranging $[0,1]$. This dampens the effect of a single dominant salient gene. In the extreme case where α is set to 0, once a virtual gene is selected all virtual genes sharing constituent genes will not be further considered. The second parameter affecting the virtual gene selection is β, which controls how likely virtual genes in the same gene cluster are selected. Different gene clusters correspond to different regulative processes in a cell. Choosing genes from different gene clusters broadens the spectrum of the selected gene set. β also ranges $[0,1]$. In the extreme situation where $\beta = 0$, only one virtual gene will be selected for each gene cluster. After modifying pairwise scores, the algorithm begins next loop to find the highest scored virtual gene. This process repeats until k virtual genes have been selected. For performance comparison of the *pairwise virtual gene* algorithm and single gene based algorithms, each *pairwise virtual gene* counts for two genes. For example, the performance of selecting 50 genes using single gene based algorithms would be compared to performance of selecting top 25 *pairwise virtual genes*.

2.4 Complexity of the Pairwise Virtual Gene Algorithm

The *pairwise virtual gene selection* algorithm runs in three stages: (1) cluster genes based on expression profile (lines 1-2), (2) calculate discriminative scores for the *pairwise virtual gene* (lines 3-11), and (3) select pairwise virtual genes with best discriminative scores (lines 12-20). We assume gene cluster number to be θ and n, m, k, α, β as discussed above.

In the first stage of analysis, k-means algorithm runs in $O(\theta n)$. In the second stage, the actual number of gene pairs examined is $O(\frac{n^2}{\theta})$, assuming gene clusters obtained in the previous stage are of roughly the same size. For each gene pair, the calculation of the pairwise virtual gene and its discriminative score require $O(m^2)$. Time complexity of the second stage is $O(\frac{m^2 n^2}{\theta})$. Stage three requires $O(k(\frac{n^2}{\theta} + m^2 + n + \frac{n}{\theta}))$ time. Putting them together, we have time complexity of $O(\theta n + \frac{m^2 n^2}{\theta} + k(m^2 + \frac{n^2}{\theta}))$. The most time consuming part in the previous expression is the term $O(\frac{m^2 n^2}{\theta})$. In our experiments, we choose $\theta \sim \Theta(n)$. Considering the fact that $k < n$, the time complexity of Algorithm 2 becomes $O(n^2 + nm^2)$. The $O(n^2)$ term is for k-means clustering, which runs rather quickly. If no clustering is performed in stage 1 (or $\theta = 1$, one gene cluster), the time complexity becomes $O(n^2 m^2 + kn^2)$. The savings in computation time is obvious.

Majority of space complexity for the *pairwise virtual gene selection* algorithm comes from stage 2 in the algorithm where pairwise discriminative scores are recorded. The space needed for that is $O(\frac{n^2}{\theta})$ using sparse array. Under typical situation if we choose $\theta \sim \Theta(n)$, space complexity of Algorithm 2 becomes $O(n)$, although with a large constant.

3 Experiments

In this section, we report extensive experimental results on three publicly available microarray datasets [1][7][18]. In each case, we study the gene selection problem in the context of two class sample classification.

3.1 Colon Cancer Dataset

Data Preparation. This dataset was published by Alon [1] in 1999. It contains measurements of expression levels of 2000 genes over 62 samples, 40 samples were from colon cancer patients and the other 22 samples were from normal tissue. The minimum value in this dataset is 5.8163, thus no thresholding is done. We perform base 10 logarithmic transformation and then for each gene, subtract mean and divide by standard deviation. We will refer to this dataset as Alon dataset in the rest of the paper.

Experiments. We performed three experiments on this dataset to evaluate performance of the four feature selection algorithms. The main purpose of each experiment is listed as follows:

1. Compare classification accuracy and the stability of classification accuracy between *single gene t-score* [3], *single gene S2N score* [1], *clustered pairwise t-score* [3] (their all pair method modified by limiting computation within gene clusters), *pairwise virtual gene*. We refer this experiment as alon.1 in this paper.
2. Study how the choice of number of clusters in the *pairwise virtual gene* algorithm affects classification accuracy and the stability of classification accuracy. We refer this experiment as alon.2 in this paper.
3. Study how the choice of initial cluster centers in the *pairwise virtual gene* algorithm affects gene selection performance. The *pairwise virtual gene* algorithm uses the k-means clustering algorithm to first divide genes into gene clusters. K-means algorithm is not stable in the sense that by supplying different initial cluster centers, different clustering results will be returned. We refer this experiment as alon.3 in this paper.

For experiment alon.1, we use three classification algorithms to measure the performance of the feature selection algorithms. The classification algorithms we use are knn (k-nearest neighbor classifier, with $k = 3$), svm (support vector machine, with radial kernel) [5] and a linear discriminant method dld (diagonal linear discriminant analysis) [17]. For cross validation of classification accuracy, we use a 2-fold cross validation method, which is the same as leave-31-out method used in [3]. We run 2-fold cross validation 100 times to obtain an estimate of classification accuracy. Standard deviation of classification accuracy is also reported here. The number of genes to be selected is limited to 100, as it is reported in the literature[1] that even top 50 genes produce good classifiers.

For experiment alon.2, we use knn with $k = 3$ as classifier. We experimented with clustering genes into $8, 16, 32, 64, 128, 256$ clusters in stage one of *pairwise virtual gene* algorithm and then measure 2-fold classification accuracy as stated in the previous paragraph.

For experiment alon.3, we use knn with $k = 3$ as classifier. Same experiments are repeated for 20 times with randomly generated initial cluster centers for stage one of *pairwise virtual gene* algorithm. Performance of our feature selection methods is reported.

In all experiments, we measure performance of selecting from 2 to 100 genes, increasing by 2 at a time. We set $\alpha = 0, \beta = 1$ in all these experiments. As stated before,

when comparing single gene-based gene selection algorithm with *pairwise virtual gene* algorithm, we treat each *pairwise virtual gene* as two genes. Thus the performance of classifiers built from top n genes are compared with the performance of classifiers built from top $\frac{n}{2}$ *pairwise virtual genes*.

Results. Results of our experiment alon.1 are summarized in Figures 2, 3, 4. In each figure, the left part plots classification accuracy against number of genes used to build classifier and the right part shows standard deviations of classification accuracy. By calculating the standard deviation, we can roughly estimate how close the mean classification accuracy is to the real classification accuracy. Each figure shows classification accuracy we archived using different classification methods.

From these experiments, we conclude that on Alon dataset, the *pairwise virtual gene* algorithm performs the best. When DLD and KNN classifiers are used, *pairwise virtual gene* algorithm is significantly better than other feature selection methods we tested. When SVM is used, all FSS methods produce comparable prediction accuracy with *pairwise virtual gene* algorithm enjoying small advantage over single gene based algorithms. The *pairwise virtual gene* algorithm is also most stable, in the sense that it has the smallest standard deviation of classification accuracy.

When testing using DLD classifier, the *pairwise virtual gene* algorithm results in 5%-10% increase in prediction accuracy over other FSS methods and almost 50% decrease in its standard deviation. The experiment with KNN classifier generates similar result with the *pairwise virtual gene* algorithm leading other FSS methods in classification accuracy by 2% and having the smallest variance.

Experiments with SVM generate more mixed results in which all four FSS methods having comparable classification accuracy. The *single gene t-score* and *single gene S2N* gene selection algorithms perform better than the *pairwise virtual gene* and *pairwise t-score* algorithms when the number of genes selected is less than 20. When more genes

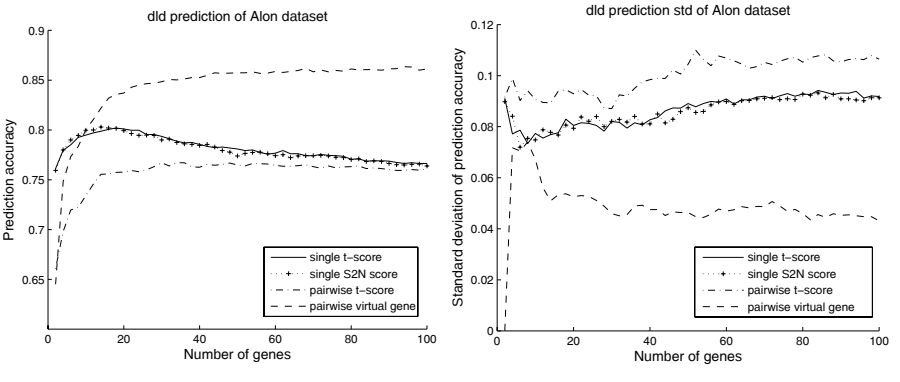


Fig. 2. Result of experiment alon.1. Prediction accuracy of four feature selection methods on Alon dataset using DLD classifier. Left figure shows prediction accuracy against the number of genes used to build DLD classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

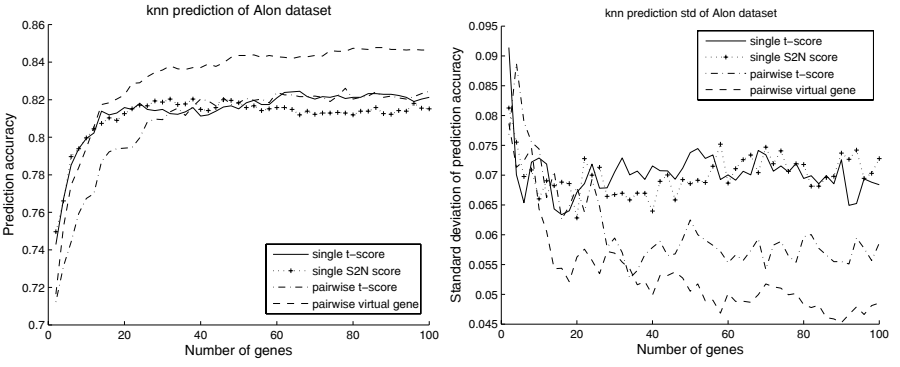


Fig. 3. Result of experiment alon.2. Prediction accuracy of four feature selection methods on Alon dataset using knn classifier (k=3). Left figure shows prediction accuracy against the number of genes used to build knn classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

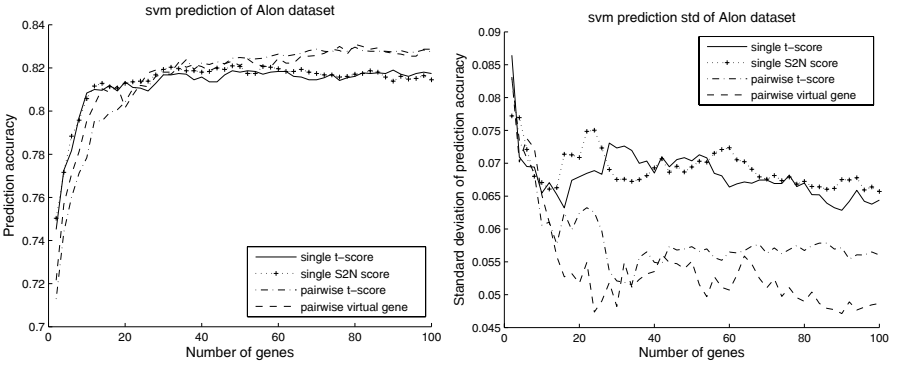


Fig. 4. Result of experiment alon.3. Prediction accuracy of four feature selection methods on Alon dataset using SVM classifier. In this experiment, we used a radial kernel for SVM. Left figure shows prediction accuracy against the number of genes used to build SVM classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

are selected, the *pairwise virtual gene* and *pairwise t-score* algorithms perform constantly better than the *single t-score* and *single gene S2N* algorithms. When the number of genes selected is more than 50, the *pairwise virtual gene* and *pairwise t-score* algorithms outperform the other two FSS algorithms by 1% in classification accuracy. The variations in classification accuracy still favors strongly towards pairwise methods with the *pairwise virtual gene* algorithm having the smallest variation.

For experiment alon.2, we measure the performance of the *pairwise virtual gene* algorithm setting the number of cluster in stage 1 of the algorithm to be $8, 16, 32, 64, 128, 256$. The results are summarized in Figure 5. We see an overall trend of decline in performance as the number of clusters increases. The classification performance peaks

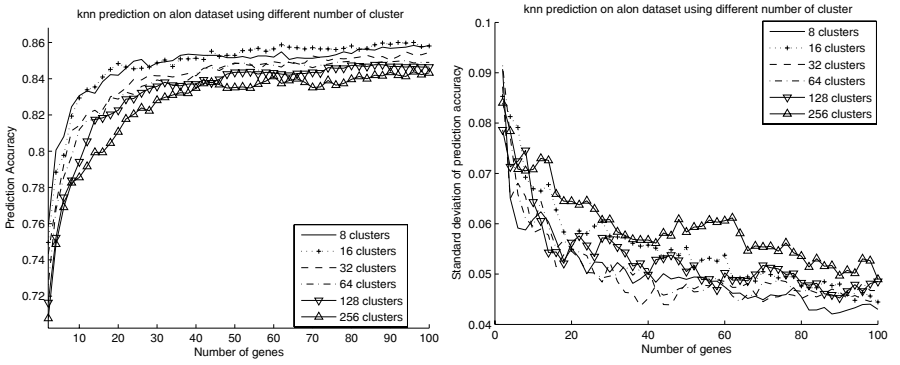


Fig. 5. Prediction accuracy and its standard deviation of knn (k=3) using different number of clusters in k-means algorithm (stage 1 of algorithm 2). Prediction accuracy degrade as the number of clusters increase. However, the within-cluster gene pairs (256-cluster version vs. 8-cluster version) retain much information as a reduction of 99.9% of pairs results only around 2% decrease in prediction accuracy.

when 8/16 clusters are used, indicating cluster numbers suitable for this dataset in that range. Compare two extremes, the 8-cluster version and the 256-cluster version, *pairwise virtual gene* algorithm performs about 2% better in classification accuracy using knn ($k = 3$) classifier when 8 clusters are used. This is somewhat we have expected since when using 256 clusters, compared to the 8-clusters version, the computed pairwise score is around $\frac{1}{32^2}$ or around 0.1%.

It is worth noting that we used a rather crude cluster analysis algorithm, the k-means algorithm. By computing only 0.1% (or omitting 99.9%) of all possible pairs in a 8-cluster version of the algorithm, we still get strong prediction accuracy, only losing about 2% of it. This indicates that correlations between genes within clusters generated by the k-means algorithm carry much more information on sample class distinction. We also expect to further improve *pairwise virtual gene* algorithm by using more sophisticated cluster analysis algorithms.

Since the k-means cluster algorithm is not stable, in the sense that initial cluster center assignments will affect clustering result, we perform experiment alon.3 to determine how the *pairwise virtual gene* algorithm is affected by it. We run 2-fold cross validation 100 times. Each time, the *pairwise virtual gene* algorithm is run 20 times with randomly generated initial gene clusters to select 20 different sets of virtual genes. The performance of 3-nn classifier using each of the 20 virtual gene sets is measured. Figure 6 plots the mean value of the classification accuracy, with its standard deviation. From this experiment, we conclude although k-means cluster algorithm is not stable, it performs well enough to capture important gene pairs. Twenty different initial cluster centers result in twenty different *pairwise virtual gene* selection. However, the final classification accuracy measured with 3-nn (3 nearest neighbor) classifier using these twenty different *pairwise virtual gene* selections does not vary much (having standard deviation of 0.3% to 0.5%). This justifies the use of the unstable k-means algorithm in our algorithm.

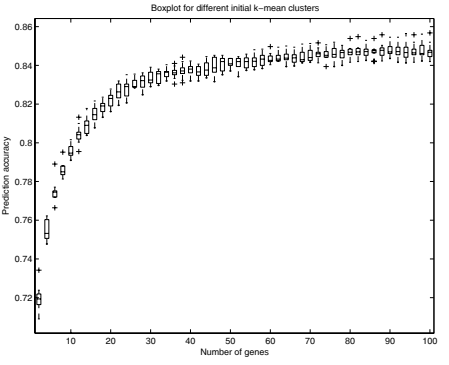


Fig. 6. The boxplot of mean 3-nn classification accuracy using *pairwise virtual gene* algorithm with 20 different initial clusters

3.2 Leukemia Dataset

Data Preparation. This dataset was published by Golub etc. [7] in 1999. It consists of 72 samples, of which 47 samples were acute lymphoblastic leukemia (ALL) and rest 25 samples were acute myeloid leukemia (AML). 7129 genes were monitored in their study. This dataset contains a lot of negative intensity values. We use the following steps (similar to Dudoit etc.[6]) preprocessing the dataset before feed to our algorithm. First we threshold the data set with floor of 1 and ceiling of 16000. Then we filter out genes with $max/min <= 5$ or $(max - min) <= 500$, where *max* and *min* are the maximum and minimum expression values of a gene. After these two steps, the resulting 3927 genes are transformed using base 10 logarithmic and then the expression levels for each gene are normalized. We will refer to this dataset as Golub dataset in the rest of this paper.

Experiments. We perform experiments to compare feature selection performance on Golub dataset. Two classifiers (KNN, DLD) are used. Classification accuracies of these classifiers using four feature selection algorithms (*single gene t-score*[3], *single gene S2N score*[7], *pairwise t-score*, *pairwise virtual gene*) are reported here. In all experiments, we measure performance of selecting from 2 to 100 genes, increasing by 2 at a time. We set $\alpha = 0, \beta = 0.8$ in all experiments.

Results. This dataset contains roughly four times the number of genes of Alon dataset. Straightforward computing of all gene pairs becomes intractable. Based on results obtained in the previous section on Alon dataset, we set the number of clusters to be 256. Results are shown in Figures 7, 8. For DLD classifier, when the number of selected genes is larger than 20, *pairwise virtual gene* algorithm performs consistently better than single gene based algorithms, though not by a large margin. For knn classifier, *pairwise virtual gene* algorithm performs consistently better than all other methods we tested. Standard deviations of the classification accuracy declines as number of genes increase with one abnormal jump for single gene based methods using DLD classifier. All feature selection methods have similar variations in the classification accuracy.

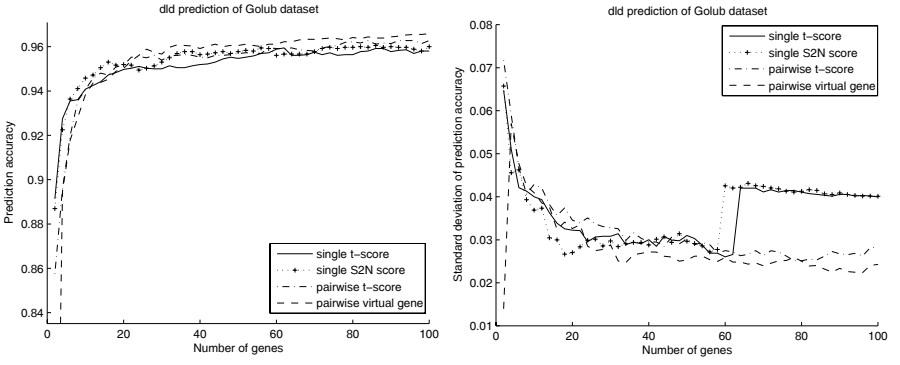


Fig. 7. Prediction accuracy of four feature selection methods on Golub dataset using DLD classifier. Left figure shows prediction accuracy against the number of genes used to build DLD classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

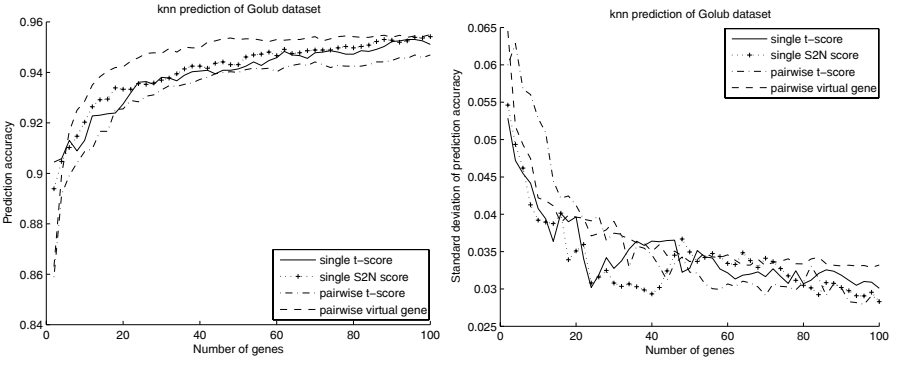


Fig. 8. Prediction accuracy of four feature selection methods on Golub dataset using knn classifier (k=3). Left figure shows prediction accuracy against the number of genes used to build knn classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

Overall, the *pairwise virtual gene* algorithm performs better than the single gene based algorithms on this dataset.

3.3 Multi-class Cancer Dataset

Ramaswamy etc. [18] reported study of oligonucleotide microarray gene expression involving 218 tumor samples spanning 14 common tumor types and 90 normal tissue samples. The expression levels of 16063 genes and expressed sequence tags were monitored in their experiments. The author separated the tumor samples into training set (144 samples) and testing set (54 samples). The rest 20 samples are poorly differentiated adenocarcinomas, which we did not include in our study. The training tumor set

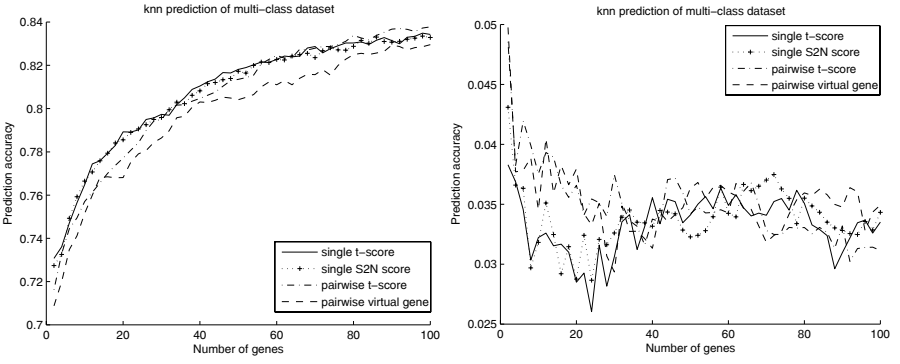


Fig. 9. Prediction accuracy of 4 feature selection methods on multi-class dataset using knn classifier (k=3). Left figure shows prediction accuracy against the number of genes used to build knn classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

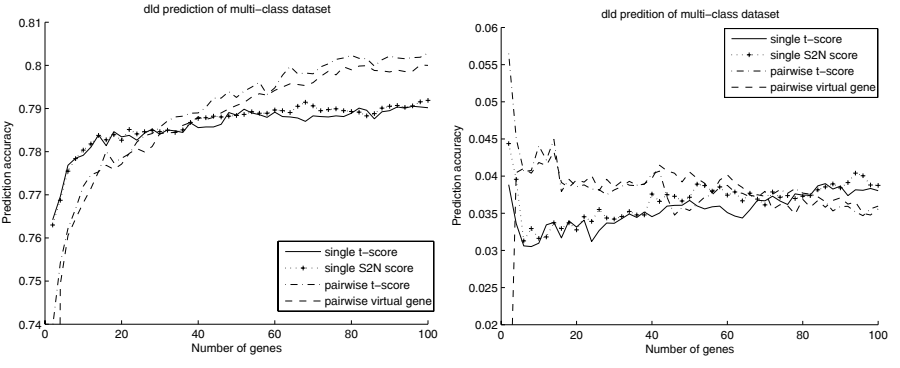


Fig. 10. Prediction accuracy of 4 feature selection methods on multi-class dataset using DLD classifier. Left figure shows prediction accuracy against the number of genes used to build knn classifier. Right figure shows the standard deviation of prediction accuracy against the number of genes.

of 144 samples and 90 normal tissue samples are combined together for our study. We refer this data set as multi-class dataset in the rest of our paper.

Data Preparation. Like the Leukemia data set, multi-class dataset contains a lot of negative values. As a data preprocessing step, we apply a thresholding of 1 and filter out genes with $max/min <= 5$ or $(max - min) <= 500$. The resulting dataset has 11985 genes. Logarithmic transformation and normalization are then performed before data are fed to gene selection algorithms. It is worth nothing that in the original paper by Ramaswamy, etc.[18] all 16063 genes (or ESTs) were used for classification. For our study of feature selection, the application of $max/min <= 5$ or $(max - min) <= 500$ filter makes sense since we are only interested in several top ranked genes.

Experiments. We measure performance of four feature selection algorithms using knn and dld classifiers. 2-fold cross validation is performed 100 times. Experiments are

in the same setting as for Alon and Golub datasets. In all experiments, we measure performance of selecting from 2 to 100 genes, increasing by 2 at a time. We set $\alpha = 0$, $\beta = 0.8$ in all experiments.

Results. In this experiment, we set the number of clusters to be used to 400. Result of knn classifier shows single gene based algorithms performs better, but within 1% of accuracy compared to *pairwise virtual gene* algorithm. *Clustered pairwise t-score* algorithm performs as good as single gene based algorithms. As the number of genes selected increases, the differences in performance gradually converge.

4 Conclusion and Future Work

Gene selection is crucial both for building a good sample classifier and for selecting smaller gene set for further biological investigation. Feature extraction algorithms (PCA, SVD, etc.), single gene based discriminative scores (t-score, S2N, TNoM, information gain, etc.) and correlation based algorithms have been proposed for this purpose. In this paper, we proposed a totally different approach. Instead of trying to minimize correlations within the selected gene set, we examined whether such correlations are good predictors of sample class labels. *Virtual gene* is a linear combination of a set of real genes. Our experiments confirm our assumption that the correlations between genes are indeed good predictors of sample class labels, better in many cases than single gene based discriminative scores. There are biological explanation for this: genes interact with each other. The relative abundance of genes is a better predictor than the absolute values. Using gene clustering algorithms to limit gene pair selection seems promising. Our experiments show that by calculating pairwise scores for only a very small portion (0.5%) of all possible gene pairs, decent classification performance can be achieved. This in turn shows most useful pairwise correlations are contained within gene clusters.

Our algorithm still has space for improvement. First but not least, we are interested in combining single gene based scores and virtual gene. In contrast to correlation based gene selection approaches, we can select top genes with high individual scores and top correlations between genes. We also want to examine larger virtual genes, virtual genes that combine more than two genes. Gene clustering is only a crude way of grouping co-regulated genes. We are currently working on using gene ontology as a way to group genes. Our algorithm is quite open, several other algorithms (e.g., cluster analysis and discriminative power of single gene) can be plugged into our algorithm without much modification. We leave this as future work as well.

References

1. U. Alon, N. Barkai, D. Notterman, K. Gish, S. Ybarra, D. Mack, and A. Levine. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissue probed by oligonucleotide arrays. *Proc. Natl. Acad. Sci. U.S.A.*, 96(12):6745–50, 1999.
2. A. Ben-Dor, L. Bruhn, N. Friedman, I. Nachman, M. Schummer, and Z. Yakhini. Tissue classification with gene expression profiles. volume 7, pages 559–83, 2000.
3. T. Bø and I. Jonassen. New feature subset selection procedures for classification of expression profiles. *Genome Biology*, 3(4):research0017.1–0017.11, 2002.

4. G. V. Bobashev, S. Das, and A. Das. Experimental design for gene microarray experiments and differential expression analysis. *Methods of Microarray Data Analysis II*, pages 23–41, 2001.
5. C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines.
6. S. Dudoit, J. Fridlyand, and T. P. Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, 97(457):77–87, 2002.
7. T. R. Golub et al. Molecular classifications of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–7, 1999.
8. I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46:389–422, 2002.
9. T. Hastie, R. Tibshirani, M. Eisen, A. Alizadeh, R. Levy, L. Staudt, W. Chan, D. Botstein, and P. Brown. 'gene shaving' as a method for identifying distinct sets of genes with similar expression patterns. *Genome Biology*, 1(2), 2000.
10. J. Jaeger, R. Sengupta, and W. L. Ruzzo. Improved gene selection for classification of microarrays. In *Proc. PSB*, 2003.
11. A. K. Jain, R. P. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.
12. D. Jiang, C. Tang, and A. Zhang. Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386, 2004.
13. J. Khan, J. Wei, M. Ringner, L. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. Antonescu, and C. Peterson. Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature Medicine*, 7(6):673–9, 2001.
14. T. Li, C. Zhang, and M. Ogihara. A comparative study of feature selection and multiclass classification methods for tissue classification based on gene expression. *Bioinformatics*, 20:2429–2437, 2004.
15. W. Li and I. Grosse. Gene selection criterion for discriminant microarray data analysis based on extreme value distributions. In *Proc. RECOMB*, 2003.
16. Y. Lu and J. Han. Cancer classification using gene expression data. *Genome Inform.*, 28:243–268, 2003.
17. K. Mardia, J. Kent, and J. Bibby. *Multivariate Analysis*. Academic Press, 1979.
18. S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. Mesirov, T. Poggio, W. Gerald, M. Loda, E. S. Lander, and T. Golub. Multiclass cancer diagnosis using tumor gene expression signatures. *PNAS*, 98(26):15149–15154, 2001.
19. V. G. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *PNAS*, 98(9):5116–5121, April 2001.
20. Y. Wang, F. S. Makedon, J. C. Ford, and J. Pearlman. Hykgene: a hybrid approach for selecting marker genes for phenotype classification using microarray gene expression data. *Bioinformatics*, 21(8):1530–1537, 2005.
21. Y. Wu and A. Zhang. Feature selection for classifying high-dimensional numerical data. In *IEEE Conference on Computer Vision and Pattern Recognition 2004*, volume 2, pages 251–258, 2004.
22. E. P. Xing, M. I. Jordan, and R. M. Karp. Feature selection for high-dimensional genomic microarray data. In *Proc. 18th International Conf. on Machine Learning*, pages 601–608. Morgan Kaufmann, San Francisco, CA, 2001.
23. L. Yu and H. Liu. Redundancy based feature selection for microarray data. In *Proc. of SIGKDD*, 2004.

Author Index