An Integrated Complexity Analysis of Problems from Computational Biology

by

Michael Trevor Hallett
B.Sc. (Honours), Queen's University at Kingston, 1992

A Dissertation Submitted in Partial Fulfillment
of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

We accept this dissertation as conforming
to the required standard

Dr. Michael R. Fellows, Supervisor (Department of Computer Science)

Dr. Valerie King, Department Member (Department of Computer Science)

Dr. Bill Wadge, Department Member (Department of Computer Science)

Dr. Ben F. Koop, Outside Member (Department of Biology)

Dr. Larry Ruzzo, External Examiner (Department of Computer Science, University of Washington, Seattle)

Supervisor: Dr. M. R. Fellows

Abstract

We perform an integrated complexity analysis on a number of combinatorial problems arising from the field of computational biology. The classic framework of $\mathcal{NP}$–completeness, algorithmic design techniques for bounded width graphs, and parameterized computational complexity together provide a clear and detailed map of the intrinsic hardness of the following problems: INTERVALIZING COLORED GRAPHS and SHORTEST COMMON SUPERSEQUENCE.

The fundamental concern of parameterized complexity is the apparent qualitative difference in algorithmic behaviour displayed by many problems when one or more input parameters are bounded. For many problems, only a small range of values for these parameters capture most instances arising in practice. This is certainly the case in computational biology in several specific arenas such as DNA physical mapping or multiple sequence alignment. At its most general level, parameterized complexity partitions problems into two classes: fixed parameter tractable (FPT) and fixed parameter intractable (hard for classes of the W–hierarchy.) The former indicates that the particular parameterization may allow for efficient practical algorithms whilst the latter indicates the parameterization is not effective (asymptotically) in alleviating the intractability.

The problem INTERVALIZING COLORED GRAPHS (ICG) models in a straightforward albeit limited way the determination of contig assemblies in the mapping of DNA. We show ICG to be $\mathcal{NP}$–complete (no polynomial time algorithm unless $\mathcal{P}$=$\mathcal{NP}$), not finite-state (a very general algorithmic design technique for bounded width graphs fails), and hard for the parameterized complexity class $W[1]$ (a specific parameterized version of ICG does not admit an efficient algorithm unless many other well-known – and apparently hard – problems admit efficient algorithms).

Both SHORTEST COMMON SUPERSEQUENCE and its sister problem LONGEST COMMON SUBSEQUENCE have applications in multiple sequence alignment. We show that SHORTEST COMMON SUPERSEQUENCE PARAMETERIZED BY THE NUMBER OF INPUT STRINGS AND THE SIZE OF THE ALPHABET is hard for complexity class $W[1]$. As is the case with ICG, this implies that it does not admit efficient algorithms unless some unlikely computational complexity collapses occur.

Examiners:

Dr. Michael R. Fellows, Supervisor (Department of Computer Science)

Dr. Valerie King, Department Member (Department of Computer Science)

Dr. Bill Wadge, Department Member (Department of Computer Science)

Dr. Ben Koop, Outside Member (Department of Biology)

Dr. Larry Ruzzo, External Examiner (Department of Computer Science, University of Washington, Seattle)

Contents

List of Tables

List of Figures

Acknowledgements

These past few years have been a lot of fun and there are many people to whom I owe Thanks.

To Ryan Hayward for giving me my first chance; to Valerie King for always expressing an (undue) amount of confidence in me, directing me towards interesting problems, sending me to conferences and, in general, always being there to talk.

To Kevin Cattell, Michael Dinneen, Patricia Evans, Lou Ibarra — it has been a pleasure working with you all. A gracious Thank You to my committee members Ben Koop, Bill Wadge, Hausi Müller, and Larry Ruzzo. To NSERC for their financial support throughout 1995 and 1996. Of course, these years would not have been the same without Todd Wareham with whom I have shared one long never-ending discussion about everything and anything.

A most special Thank You goes out to Sister Cristina Tognon (for the sanctuary she provided), Sister Rosa Mastri (just for being you), Brother Mark Robertson (those guitar "lessons" and sharing your books) and the Reverend Jason Rodney (fellow matinee film critic and cultural advisor). It goes without saying but "Buy Knockin' Dog". To Anne, Wes, Mindy, Eliza and Rob. Thank You.

A loving Thank You to my mom and dad, Wesley, Kelly and grandparents who have supported me these last few decades.

A warm and heartfelt Thank You to Helen Pritchard who has been with me this past year. I have appreciated all that you have done for me and I'm looking forward to our time ahead. It just wouldn't be the same without you in my life.

Of course, a huge Thank You to my friend (and supervisor) Michael Fellows for the fun, the funding, the surf, conversations, beer, encouragement, for letting me go so graciously and taking me back so easily, the dinners, the conferences, the advice, and – well, just about everything ...

Chapter 1

Introduction

The Human Genome project and other similar efforts have as their mandate the mapping of entire genomes for a few select organisms and the development of technologies that will allow for the expeditious and routine mapping of large numbers of diverse organisms. Current technologies for deducing these maps introduce several computational problems. We consider the SEQUENCE RECONSTRUCTION problem which arises in the determination of restriction–enzyme cutting sites, nucleotide sequencing and the identification of polymorphic DNA markers. The ultimate goal of this problem is to reconstruct a sequence given a set of overlapping subfragments from this sequence. We model this combinatorially with the problem

> INTERVALIZING COLORED GRAPHS (ICG)
> **Instance:** A graph $G = (V, E)$ and a coloring $c : V \to \{1 : k\}$.
> **Question:** Is there a supergraph $G' = (V, E')$ of G which is properly colored by c and which has an interval representation $\mathcal{I}$?

Intuitively, this problem seeks to find a superset of the edges with the property that no two like–colored vertices are adjacent and the graph can be "laid out" in a linear fashion: that is, the interval for each vertex can be placed along a linearly ordered set so that two intervals overlap if and only if there is an edge between the corresponding vertices in the graph.

We perform an integrated complexity analysis of ICG from a number of distinct frameworks. The *classical* theory of $\mathcal{NP}$–completeness. parameterized computational complexity and several general algorithmic design methodologies for bounded width graph families fit

together to give a clear and detailed perspective of the computational difficulty of this problem.

The SEQUENCE COMPARISON problem is used by molecular biologists and biochemists to identify regions from a set of molecular sequences having a high degree of similarity. Similarity may indicate shared attributes or phylogenetic relationships between the organisms these sequences represent. If they represent homologous genes, it may be possible to deduce their structure or function from this information. At an abstract level, the SEQUENCE COMPARISON problem may be viewed as string matching problems. We consider

> SHORTEST COMMON SUPERSEQUENCE (SCS).
> **Instance:** Alphabet Σ, set of strings $R = \{r_1, r_2, \ldots, r_k\} \in \Sigma^*$, integer $M \in \mathbf{N}$.
> **Question:** Does there exist a string $S \in \Sigma^*$ of length at most M, that is a supersequence of each string in R?

Intuitively, this problem seeks to find the shortest possible string S such that a given set of input strings are all subsequences of it.

The literature contains an overwhelming number of "negative" results pertaining to SCS (and related string matching problems). For instance, SCS is $\mathcal{NP}$ and $\mathcal{MAXSNP}$-hard when restricted to a binary alphabet. These results provide great evidence that neither *fast* exact algorithms nor *fast* approximation schemes exist for it. Given the apparent difficulty of these problems, biologists and biochemists are constrained to measuring the similarity of a small number of sequences at a time. Furthermore, the *number* of such sequences is independent of their *length* and are formed over a small bounded size alphabet. These observations provide a possible avenue for coping with the intractability: it may be possible to design algorithms with running times exponential only in the number of strings and the size of the alphabet but polynomial in the length of the strings. Parameterized computational complexity allows one to address this issue and with this tool we examine the problem SCS when restricted to a small number of input sequences formed over a restricted alphabet size.

This thesis applies computational complexity frameworks to two problems arising from computational molecular biology. These tools allow one to identify problems unlikely to have *fast* algorithms and to cope with this intractability. Defining precisely what is meant by *fast* is difficult. Classically, a problem is said to admit an efficient or fast solution if there exists an algorithm which requires at most a polynomial number of steps in the size of the input on all instances of that problem. We duly note that there are certain degenerate

behaviours allowed under this liberal definition. We will address this issue in more depth in subsequent chapters.

The primary means of identifying problems unlikely to have fast algorithms is via the *completeness* framework. Certainly, the most famous such framework centers around the classes $\mathcal{P}$ and $\mathcal{NP}$ of which [54] contains a thorough examination. In some sense, this framework has become the *classical* approach to computational complexity. Showing a problem hard for $\mathcal{NP}$ implies one must look towards alternative means for solving the problem at hand. Dynamic programming, "branch and bound" methods and approximation algorithms are several such techniques [54]. The parameterized computational complexity framework introduced by Downey and Fellows in [38, 42] offers a markedly different approach to *coping with intractability.*

The fundamental concern of parameterized complexity is the apparent qualitative difference in algorithmic behaviour displayed by many problems when one or more input parameters are bounded. For many problems, only a small range of values for these parameters capture most instances arising in practice. This is certainly the case in computational biology in several specific arenas such as DNA physical mapping or multiple sequence comparison. At its most general level, parameterized complexity partitions problems into two classes: fixed parameter tractable (FPT) and fixed parameter intractable (hard for W). The former indicates that the particular parameterization may allow for efficient practical algorithms whilst the latter indicates the parameterization is not effective (asymptotically) in alleviating the intractability.

It should be noted that parameterized computational complexity is not merely a refinement of classical complexity. As alluded to before, definitions of *fast* (*tractable*) such as the classical notion of polynomial time ($\mathcal{P}$–time) allow for certain types of degenerate behaviours as do parameterized notions of tractability within the class FPT. Analyses from both frameworks provide two different and complementary perspectives of a problem.

Sections 2.1 and 2.2 of Chapter 2 provide a detailed explanation of these frameworks. This chapter also contains a section describing various general algorithmic design methods associated with the parameterized class FPT.

In Chapter 3, we show ICG to be $\mathcal{NP}$-complete (no polynomial time algorithm unless $\mathcal{P}=\mathcal{NP}$), finite-state for the case where the number of colors, k, is 2, (a very general algorithmic design technique for bounded width graphs yields a fast recognition algorithm) and not finite-state when $k \geq 3$ (this technique fails for these cases). Furthermore, we

show ICG is hard for the parameterized complexity class $W[1]$ (a specific parameterized version of ICG does not admit an efficient algorithm unless many other well-known – and apparently hard – problems admit efficient algorithms).

In Chapter 4, we show that SHORTEST COMMON SUPERSEQUENCE PARAMETERIZED BY THE NUMBER OF INPUT STRINGS AND THE SIZE OF THE ALPHABET is hard for the complexity class $W[1]$. As with ICG, this implies that it does not admit an efficient algorithm unless some unlikely computational complexity collapses occur.

Chapter 5 summarizes the major contributions of this work and lists a number of open problems.

Chapter 2

Background and Tools

The following three sections provide the necessary tools and frameworks used in latter chapters. In Sections 2.1 and 2.2, we provide a brief introduction to classical and parameterized computational complexity which are used throughout Chapter 3 and Chapter 4 in our examination of problems INTERVALIZING COLORED GRAPHS and SHORTEST COMMON SUPERSEQUENCE. Subsection 2.3.1 presents definitions for *treewidth* and *pathwidth* and Subsections 2.3.2 and 2.3.3 build upon this notion of bounded width giving overviews of two useful tools which, when applied to problems, act as a sort of *litmus* test. Passing either of these tests implies the existence of very fast recognition algorithms.

We assume the reader has a basic knowledge of graph theory and formal language theory. It would be ideal if the reader had a cursory understanding of complexity theory.

2.1 Classical Computational Complexity

In the interests of thoroughness, we sketch the framework surrounding the now classical notion of $\mathcal{NP}$-completeness theory. Our definitions and notation follow closely that used in [54]. Any deviations from this style will be noted.

We are concerned with the *decision* version of a problem Π; that is, a problem which asks a "yes" or "no" question. Generally, it is easy to formulate a decision version given an optimization problem and vice versa. A decision problem Π consists of a set of *instances*, D_Π, and a subset of "yes" instances $Y_\Pi \subseteq D_\Pi$.

Let

$$L[\Sigma, e] = \left\{ x \in \Sigma^* : \begin{array}{l} e \text{ is any polynomially related encoding scheme of} \\ I \in Y_\Pi \text{ over the alphabet } \Sigma \end{array} \right\}$$

where ***polynomially related*** in this context means that, for any scheme e for Π, there exists two polynomials p and p' such that if $I \in D_\Pi$ and x is a string encoding an instance I under e, $|I| \leq p(|x|)$ and $|x| \leq p'(|I|)$, where $|\cdot|$ denotes the length of a string and p is a polynomial function.

Where no confusion may arise, we simply write L instead of $L[\Sigma, e]$.

A deterministic Turing machine M with input alphabet Σ *accepts* $x \in \Sigma^*$ if and only if M halts in state q_y when applied to input x. The language L_M *recognized* by the machine M is given by $L_M = \{x \in \Sigma^* | M \text{ accepts } x\}$.

For the remainder of this paper, complexity is measured solely in terms of *time*. Let the *time complexity function* $T_M : \mathbf{Z}^+ \to \mathbf{Z}^+$ be given by

$$T_M(n) = max \left\{ m : \begin{array}{l} \text{there is an } x \in \Sigma^*, \text{ with } |x| = n, \text{ such that the computation of} \\ M \text{ on input } x \text{ takes time } m \end{array} \right\}.$$

If there exists a polynomial function p such that, for all $n \in \mathbf{Z}^+$, $T_M(n) \leq p(n)$, then M is called a *polynomial time deterministic Turing machine program.*

Central to the notion of ***tractability*** in classical complexity is the class

$$\mathcal{P} = \left\{ L : \begin{array}{l} \text{there is a polynomial time deterministic Turing machine} \\ \text{program M for which } L = L_M. \end{array} \right\}.$$

By allowing a polynomially bounded amount of nondeterminism in a Turing machine, we arrive at a class called $\mathcal{NP}$ widely conjectured to properly contain $\mathcal{P}$. We begin by defining a new time complexity function modified to accommodate this nondeterminism. Let $T_M : \mathbf{Z}^+ \to \mathbf{Z}^+$ be given by

$$T_M(n) = max \left\{ \{1\} \cup \left\{ m : \begin{array}{l} \text{there is an } x \in L_M \text{ with } |x| = n \text{ such} \\ \text{that the time to accept } x \text{ by } M \text{ is } m \end{array} \right\} \right\}$$

$$\mathcal{NP} = \left\{ L : \begin{array}{l} \text{there is a polynomial time nondeterministic Turing machine program} \\ M \text{ for which } L_M = L \end{array} \right\}.$$

A result showing $\mathcal{P} = \mathcal{NP}$ would imply that no additional computing power is gained by allowing a Turing machine to use a polynomially bounded amount of nondeterminism. The notion of *reductions* allows the comparison of the relative difficulty of deciding two languages.

Definition 1 ($\mathcal{P}$–time reduction) A $\mathcal{P}$–time reduction from a language $L_1 \subseteq \Sigma_1^*$ to a language $L_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \to \Sigma_2^*$ that satisfies the following two conditions: (1) there is a polynomial time DTM program that computes f and (2) for all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

Note that a reduction from a decidable language L to a decidable language L' indirectly provides an algorithm for L if we know an algorithm for L'. Furthermore, if $L' \in \mathcal{P}$ and the reduction is in fact a $\mathcal{P}$–time reduction, then L is also in $\mathcal{P}$. It is easy to prove that these reductions are transitive; that is, if L $\mathcal{P}$–time reduces to L' and L' $\mathcal{P}$–time reduces to L'', then L $\mathcal{P}$–time reduces to L''. In order to "separate" the hard languages from the easy languages within this large class $\mathcal{NP}$, we must isolate the most difficult language. The following class is the set of $\mathcal{NP}$ languages reducible from this hardest language.

Definition 2 ($\mathcal{NP}$–completeness) A language L is $\mathcal{NP}$–complete if (1) $L \in \mathcal{NP}$ and (2) L is $\mathcal{NP}$–hard: that is, L' $\mathcal{P}$–time reduces to L, where L' is known to be $\mathcal{NP}$–complete.

Of course, what is needed is the first $\mathcal{NP}$–complete problem from which to reduce *from*. S. Cook in 1971 showed the following was indeed equivalent to the hardest problem in $\mathcal{NP}$ [32].

SATISFIABILITY
Instance: A propostional formula Φ , variable set $X = \{x_1, x_2, \ldots, x_m\}$.
Question: Does there exist a truth assignment to X satisfying formula Φ?

Theorem 1 (Cook's Theorem [32]) SATISFIABILITY is $\mathcal{NP}$–complete. ∎

Since Cook's Theorem, literally hundreds of problems have been shown $\mathcal{NP}$–complete. See [54] for a more thorough description of this framework.

2.2 Parameterized Computational Complexity

Parameterized computational complexity, introduced by Downey and Fellows [1, 39, 40, 41, 42, 43], is founded on the observation that the overwhelming majority of problems take as input two or more parameters (see [54]). They are concerned with languages $L \subseteq \Sigma^* \times \Sigma^*$ and if $\langle x, k \rangle$ is in a parameterized language L, we call k the parameter. In the interests of readability and with no loss of generality to the theory, we will assume that the parameter k has domain N: that is, $L \subseteq \Sigma^* \times \mathbf{N}$. For fixed k, we call $L_k = \{x | \langle x, k \rangle \in L\}$ the k-th *slice* of L.

The primary intention is to study languages that are *tractable* "by the slice". This theory was motivated by the observation that for many problems from areas such as VLSI design, only a small range of values for some input parameters capture most instances arising in practice.

Definition 3 (FPT) For a parameterized language L, we say that L is *(uniformly) fixed parameter tractable* (FPT) if there exists a constant α and an algorithm Φ such that Φ decides if $\langle x, k \rangle \in L$ in time $f(k)|x|^\alpha$ where $f : \mathbf{N} \rightarrow \mathbf{N}$ is an arbitrary function.

The following problem

> VERTEX COVER
> **Instance:** A graph $G = (V, E)$, an integer k.
> **Parameter:** k.
> **Question:** Does there exist a set $V' \subseteq V$, $|V'| = k$, such that, for each edge $(u, v) \in E$, $u \in V'$ or $v \in V'$?

is the canonical example of membership in FPT. The following is the best upper bound known.

Theorem 2 ([9]. **Also [25, 42].**) VERTEX COVER is decidable in time $O(k|V| + (4/3)^k k^2)$. ∎

Many other languages have been shown to belong to FPT including CUTWIDTH, FEEDBACK VERTEX SET, GATE MATRIX LAYOUT, MAX LEAF SPANNING TREE, SEARCH NUMBER, TREEWIDTH, HEREDITARY PROPERTY GRAPH MODIFICATION and PATHWIDTH (see [3, 30, 42, 43]). Slightly altering the definition of *uniform* FPT above by allowing an algorithm Φ_k

for each slice of the problem instead of a single algorithm Φ for all k, yields the class *non-uniform FPT*. GRAPH LINKING NUMBER [51, 76] is an example of a problem belonging to this class and not known to belong to *uniform FPT*. See [42, 43, 59] for further examples. Unless stated otherwise all references to the class *FPT* refer to *uniform FPT*.

The algorithmic behaviour exhibited by VERTEX COVER provides sharp contrast to that displayed by the problem

SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION
Instance: A nondeterministic Turing machine M operating on alphabet Σ, a word $x \in \Sigma^*$, a positive integer k.
Parameter: k.
Question: Is there a computation of M on x that reaches an accept state in $\leq k$ steps?

Note that without the parameter k, this problem is undecidable and if the range of k is as large as the number of states of M, this problem is $\mathcal{NP}$-complete. The best known algorithms require time $\Omega(n^{f(k)})$ with $f(k) \to \infty$ and are essentially no more sophisticated than trying all of the $O(n^k)$ possible computation paths for M where n is the number of states in M. (see Figure 2.1). Intuitively, this is our notion of hardness in the parameterized setting. Note that the running time has a *factor of* k in the degree of the bounding polynomial and, when k is fixed, the algorithm is polynomial but still considered intractable. The reader should take notice of the disparity between parameterized notions of tractability and those found in the more classical approach to complexity ($\mathcal{P}$ and $\mathcal{NP}$).

Behind most, if not all, completeness frameworks lie two main ingredients. The first of which is the reduction and the second is the identification of large classes of intractable languages.

2.2.1 Parameterized Reductions

We use the following as our basic working definition but note that there are more general versions of parameterized reductions which allow for nonuniformity and nonrecursivity in the bounding functions (see [40, 43]).

Definition 4 (Parameterized many:1 reductions) We say that L reduces to L' by a standard parameterized m-reduction if there is an algorithm Φ which transforms $\langle x, k \rangle$ into

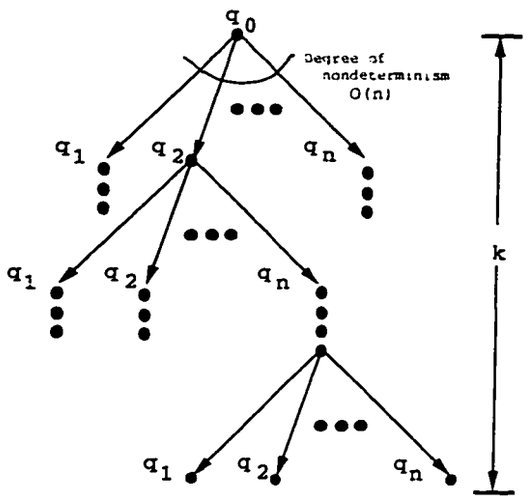

Figure 2.1: Illustration of the $O(n^k)$ computation paths for SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION, where n is the number of states in M and q_0 is the initial state.

$\langle x', g(k)\rangle$ in time $f(k)|x|^\alpha$, where $f, g : \mathrm{N} \to \mathrm{N}$ are arbitrary functions and α is a constant independent of k, so that $\langle x, k\rangle \in L$ if and only if $\langle x', g(k)\rangle \in L'$.

Note that whenever a parameterized language L parametrically reduces to L', if L' is fixed parameter tractable, then so too is L. We say that L' is at least as hard as L if L parametrically reduces to L'. It is easy to see that these reductions are transitive; that is, if L parametrically reduces to L' and L' parametrically reduces to L'', then L parametrically reduces to L''.

2.2.2 Intractability and Complexity Classes

The second necessary ingredient for our parameterized framework is the identification of large classes of intractable problems. Classical complexity begins with the problem SATISFIABILITY and Cook's theorem shows that this problem is equivalent to the hardest problem in $\mathcal{NP}$ (see Section 2.1). Likewise, the parameterized framework is built upon a similar yet more refined analogue of this basic problem from logic. It is helpful to view these logical problems as Boolean circuits with a single output gate. Each variable in the propositional formula corresponds to an input line of the circuit. A variable is assigned *true* if and only if the corresponding input line is assigned the value 1. Before proceeding, we offer the

following definitions.

Definition 5 (Decision Circuit) A *decision* circuit C is a circuit with exactly one output line. Given $x \in \{0,1\}^n$, an input vector to the n input lines of C, $C(x)$ is said to be true if the output line has the value 1. Otherwise, it is said to be false.

Definition 6 (Circuit Definitions) A Boolean circuit is of *mixed type* if it consists of circuits having gates of two kinds:

1. Small gates: *not* gates, *and* gates and *or* gates with some predetermined bound on the fan-in.

2. Large gates: *and* gates and *or* gates with unrestricted fan-in.

A Boolean circuit has fan-out one.

Definition 7 (Depth and Weft) The *depth* of a circuit C is defined to be the maximum number of gates (small or large) on an input-output path in C. The *weft* of a circuit C is the maximum number of large gates on an input-output path.

Definition 8 (Bounded Depth, Bounded Weft, Weight) A family of decision circuits F has *bounded depth* if there is a constant h such that every circuit in the family F has depth at most h. F has *bounded weft* if there is a constant t such that every circuit in the family F has weft at most t. The *weight* of a Boolean vector x is the number of 1s in the vector (the Hamming weight).

Let $F = \{C_1, C_2, \ldots C_i, \ldots\}$ be a family of decision circuits. Associated with F is a parameterized language

$$L_F = \{< C_i, k > | C_i \text{ has a weight } k \text{ satisfying assignment } \}.$$

Definition 9 (Membership for $W[t]$) A parameterized problem L belongs to $W[t]$ if L reduces to the parameterized problem $L_{F(t,h)}$ for the family $F(t,h)$ of mixed type circuits of weft at most t and depth at most h, for some constant h.

Definition 10 (Membership for $W[SAT]$ and $W[P]$) A parameterized problem L belongs to $W[SAT]$ if L reduces to the parameterized problem L_F where F is the set of all circuits with no restrictions on the depth or weft and every gate has fan-out 1. If we remove the restriction on the fan-out of the gates, L belongs to $W[P]$.

These definitions induce the following hierarchy of parameterized classes

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq \ldots \subseteq W[SAT] \subseteq \ldots \subseteq W[P]$$

and it is conjectured that the inclusions are proper. Table 2.1 lists many of the known hard or complete problems (see [43, 59] for a more complete list).

It is easy to verify the following theorem by the above definitions.

Theorem 3 ([40, 43]) For any problem L complete for $W[t]$, for any t, L is fixed parameter tractable if and only if every problem in $W[t]$ is fixed parameter tractable. ∎

We are now ready to state the parameterized analogues of SATISFIABILITY.

> WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (t,h–WCSAT)
> **Instance:** A weft t depth h circuit C, an integer k.
> **Parameter:** k.
> **Question:** Does C have a weight k satisfying assignment?

If we remove the weight k condition from the above definition, CNF–SATISFIABILITY is subsumed by WEFT 2 DEPTH 2 CIRCUIT SATISFIABILITY. Intuitively, the level of the W-hierarchy where a language L resides is determined by the weft of the circuit needed to check a candidate solution. Whereas classical complexity places all $\mathcal{NP}$-hard, polynomial time checkable languages at the same degree, the parameterized framework places a language at the degree corresponding to its "natural logical depth".

It is proven in [40] that transforming a weft t depth h circuit into an equivalent but *well-organized* circuit is fixed parameter tractable. By well organized, we mean a circuit where

- *not* gates only appearing directly below input lines,
- all small gates appear before any large gate,

- there are t alternating rows of large *and* gates and large *or* gates in products-of-sums-of-products-of- ... form, and

- the circuit has a single output line.

Similarly, but in the domain of logic, we define a t-normalized propositional formula as follows.

Definition 11 (t-Normalized) We say that a propositional formula Φ is *t-normalized* if Φ is of the form products-of-sums-of-products... of literals with t-alternations.

The following problem serves the practical role of providing a conceptually easy problem to reduce *from*. The theorem following the definition shows that it indeed captures the entire class $W[t]$.

WEIGHTED t-NORMALIZED SATISFIABILITY
Instance: A t-normalized Boolean expression Φ, a positive integer k.
Parameter: k.
Question: Does Φ have a satisfying truth assignment of weight k?

Theorem 4 (The Normalization Theorem [40]) For all $t \geq 1$, WEIGHTED t-NORMALIZED SATISFIABILITY is complete for $W[t]$. ∎

For the remaining two classes, $W[SAT]$ and $W[P]$, we define the following two problems which are straightforward extensions of WEIGHTED t-NORMALIZED SATISFIABILITY.

WEIGHTED SATISFIABILITY
Instance: A Boolean formula Φ, an integer k.
Parameter: k.
Question: Does Φ have a weight k satisfying assignment?

WEIGHTED CIRCUIT SATISFIABILITY
Instance: A decision circuit C, an integer k.
Parameter: k.
Question: Does C have a weight k satisfying assignment?

By definition, these problems are complete for $W[SAT]$ and $W[P]$ respectively (see [1, 2]).

$W[P]$	LINEAR INEQUALITIES	[1, 2]
	MINIMUM AXIOM SET	[1, 44]
	SHORT SATISFIABILITY	[1, 2]
	WEIGHTED CIRCUIT SATISFIABILITY	[1, 2]
	•	
	•	
	•	
$W[SAT]$	WEIGHTED SATISFIABILITY	[1, 2]
	•	
	•	
	•	
	LONGEST COMMON SUBSEQUENCE (k, $\|\Sigma\|$) (hard)	[15]
	COLORED CUTWIDTH (hard)	[17, 18]
$W[t]$,	FEASIBLE REGISTER ASSIGNMENT (hard)	[17, 18]
for all t	TRIANGULATING COLORED GRAPHS (hard)	[17, 18]
	BANDWIDTH (hard)	[17, 19]
	PROPER INTERVAL GRAPH COMPLETION (hard)	[17, 18]
	INTERVALIZING COLORED GRAPHS (hard)	[17, 18]
	WEIGHTED t–NORMALIZED SATISFIABILITY	[40]
	•	
	•	
	•	
$W[2]$	WEIGHTED $\{0,1\}$ INTEGER PROGRAMMING	[40]
	DOMINATING SET	[40]
	SHORTEST COMMON SUPERSEQUENCE (k)(hard)	[48]
	SHORT POST CORRESPONDENCE	[28]
	WEIGHTED q–CNF SATISFIABILITY	[41]
$W[1]$	VAPNIK–CHERVONENKIS DIMENSION	[37, 42]
	LONGEST COMMON SUBSEQUENCE (k, M)	[16]
	INDEPENDENT SET	[41]
	CLIQUE	[41]
	SHORT NTM COMPUTATION	[28]
	FEEDBACK VERTEX SET	[42]
FPT	GRAPH GENUS	[50, 51, 77]
	MINOR ORDER TEST	[76]
	GATE MATRIX LAYOUT	[50]
	TREEWIDTH	[12]
	VERTEX COVER	[9, 25, 42]

Table 2.1: A partial list of problems known to be hard or complete for the W-hierarchy. The double line demarcates those problems conjectured to be inherently intractable ($W[1]$, $W[2]$, ...) and those known to be fixed parameter tractable (FPT).

	Classical	Parameterized
VAPNIK-CHERVONENKIS DIMENSION	$\mathcal{LOGSNP}$-complete [a]	W[1] complete
VERTEX COVER	$\mathcal{NP}$-complete	FPT
CLIQUE	$\mathcal{NP}$-complete	W[1] complete
ALTERNATING HITTING SET	$\mathcal{PSPACE}$-complete	FPT
POST CORRESPONDENCE	$\mathcal{UNDECIDABLE}$	W[1]-complete

Table 2.2: A comparison of the *classical* complexity versus the parameterized complexity of several problems.

[a] Believed not to be $\mathcal{NP}$-hard.

2.2.3 Connections and Philosophy

Parameterized complexity is not simply a refinement of classical complexity. The working hypothesis states *the parameterized complexity of a problem bears no relation to its classical complexity*. Several problems witness this statement (see Table 2.2). However, there do exist some connections between these two frameworks. The simplest example is the following:

Theorem 5 ([40]) If $\mathcal{P}=\mathcal{NP}$, then $W[P] = FPT$. ∎

Thus a proof that SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION is not FPT would imply $\mathcal{P} \neq \mathcal{NP}$. The converse is unknown and there are reasons to suspect that it is not true; although, there is the following "quantitative" result.

Theorem 6 ([39]) If $W[P] = FPT$, then CIRCUIT SATISFIABILITY is solvable time $O(2^{o(v)}p(n))$ where n is the size of the circuit C and v is the number of input variables to C. ∎

For several other more obscure relations, see [39].

Parameterized reductions tend to be technically more challenging than their more familiar counterparts, the Cook or Karp (*many* : 1) $\mathcal{P}$-time reductions from classical complexity. A partial explanation for this difference in difficulty is found in [43] (see also [31]). Whereas $\mathcal{P}$-time Karp reductions are Σ_2-complete, indexed sets of uniform parameterized reductions are Σ_3-complete when the bounding function f is recursive. When f is non-recursive, they are Σ_4-complete.

There are several other characterizations of the parameterized hierarchy. For instance, [43, 45] characterize it in terms of parameterized circuit theory. In [29], the class FPT is

shown equivalent to the class of problems running in polynomial time and using $f(k)$ bits of advice for each slice k. Also, in [26, 27, 29], the W classes are re-expressed in terms of a class called $\mathcal{GC}$ or "Guess and Check" and the tight relationship between the parameterized hierarchy and alternating Turing machines is explicated.

2.3 Algorithmic Design Methodologies inside FPT

There is an interesting dialectic between complexity theory and algorithmic design. Knowing a problem belongs to a certain class can, in many cases, help guide the intuition of the algorithm designer towards a solution. The development of parameterized complexity and, in particular, the class FPT has yielded some very general design principles. Downey and Fellows [42, 43] note that most problems known to be in FPT have algorithms which fall under one of the following rubrics.

1. Method of bounded search trees

2. Reduction to a problem kernel

3. Well–quasiordering and the Robertson–Seymour theorems

4. Monadic second order logic and Courcelle's theorem

5. Cutset regularity for boundaried graphs

Of central importance to methods 3–5 is the notion of bounded width graphs. In the next subsection we formally define *pathwidth* and *treewidth* and detail how these may be used in conjunction with finite automata and logic to provide a general method of demonstrating tractability to a wide class of properties for graphs.

2.3.1 Bounded Width Graphs

Originally introduced by Robertson and Seymour in their seminal papers on graph minors [76, 77], *treewidth* provides a measure of just how tree-like a graph is.

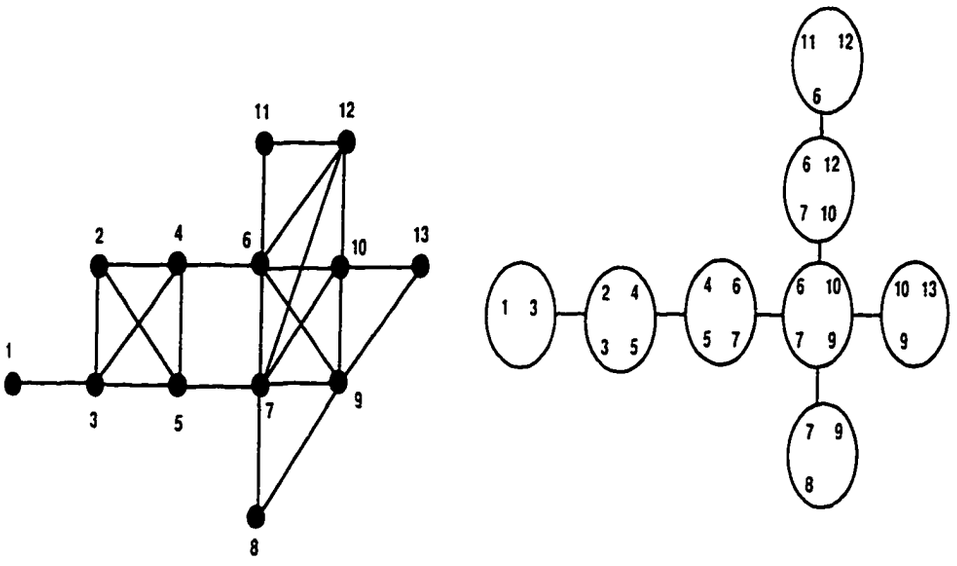

Figure 2.2: A graph of treewidth 3 and a corresponding decomposition.

Definition 12 (Treewidth) A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of V, and $T = (I, F)$ a tree, such that

- $\bigcup_{i \in I} X_i = V$.
- For all $(v, w) \in E$, there exists an $i \in I$ with $v, w \in X_i$.
- For all $v \in V$, $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T.

The *width* of a tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is $\max_{i \in I} |X_i| - 1$. The treewidth of a graph is the minimum width over all possible tree-decompositions of that graph.

Similarly, *path-decomposition* provides a measure on just how path-like a graph is. See Figures 2.2 and 2.3 for examples of graphs and their corresponding decompositions.

Definition 13 (Pathwidth) A tree-decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is a *path-decomposition*, if T is a path. The pathwidth of a graph is the minimum width over all possible path-decompositions of that graph.

Definition 14 (Bounded Width) A family $\mathcal{F}$ of graphs has *bounded treewidth* (*bounded pathwidth*) if there is some k such that for all $G \in \mathcal{F}$, G has treewidth$\leq k$ (pathwidth $\leq k$).

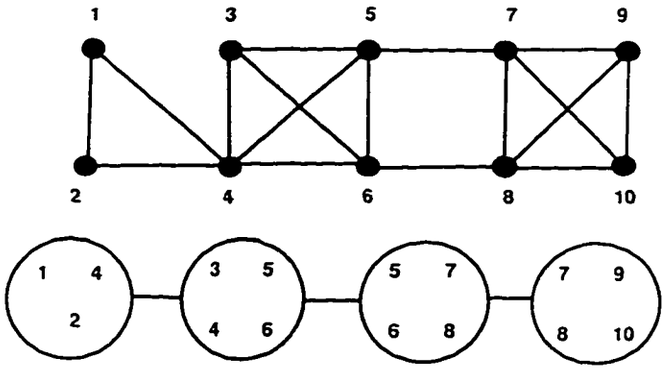

Figure 2.3: A graph of pathwidth 3 and a corresponding decomposition.

Folklore says *most graph problems are $\mathcal{NP}$-hard but most parameterized variants become easy for bounded width.* The following well-known problems are all examples of this phenomenon: FEEDBACK VERTEX SET, SEARCH NUMBER, GATE MATRIX LAYOUT and CUTWIDTH [43, 59].

For a graph family known to have a natural treewidth (pathwidth) bound, the general procedure is as follows. Using the algorithm of Bodlaender [12], in linear time we are given a tree decomposition (path decomposition) of width $b(k)$ or told that none exists. In the latter case, we stop answering "no". Otherwise, there are several ways in which to exploit this *tree* or *path* decomposition. One may simply create an ad hoc algorithm of which there are many examples (see [6, 8, 11, 13, 14, 68, 78, 81] and references found therein). The second is to show that the bounded width graph family $\mathcal{F}$ is *finite state*: that is, recognizable by a finite state automaton. There are at least two methods of achieving this. It is sufficient to show that $\mathcal{F}$ is expressible in the second order monadic logic of graphs by a result due to Courcelle. Alternatively, we can show that the graph family is what Abrahamson and Fellows term *cutset regular* [3].

2.3.2 Second-order monadic logic of graphs and Courcelle's theorem

We begin by defining the second order monadic logic of graphs.

Definition 15 (Second-order monadic logic of graphs) The syntax of second-order monadic logic of graphs includes the following logical connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$, variables for vertices, edges, sets of vertices and sets of edges, the quantifiers $\forall$, $\exists$ that can be applied to these variables, and the five binary relations:

- $u \in U$ where u is a vertex variable and U is a vertex set variable.

- $d \in D$ where d is an edge variable and D is an edge set variable.

- $inc(d, u)$ where d is an edge variable, u is a vertex variable, and the interpretation is that the edge d is incident on the vertex u,

- $adj(u, v)$ where u and v are vertex variables and the interpretation is that u and v are adjacent vertices, and

- equality for vertices, edges, sets of vertices and sets of edges.

In the interests of readability, we denote vertex and edge variables with lower case letters and reserve upper case letters for representing sets of vertices or sets of edges.

Theorem 7 (Courcelle's Theorem [34]) Let t be the width bound on a graph family $\mathcal{F}$. If $\mathcal{F}$ is a family of graphs described by a sentence in second-order monadic logic, then $\mathcal{F}$ is finite-state. ∎

Showing a graph family is expressible in second-order monadic logic implies the graph family is finite-state; that is, recognizable by a finite-state automaton. Courcelle's theorem does provide a constructive means of deducing the automaton. The main issue in this construction is efficiency since current methods require time exponential in t, the tree or path width bound. There is some hope that a fully automated software system will soon be available which will produce the appropriate finite automaton for a family $\mathcal{F}$ given only a description of $\mathcal{F}$ in the second-order monadic logic of graphs [46].

In [24], Borie, Parker and Tovie independently prove a similar result on the automatic generation of algorithms from predicate calculus.

2.3.3 Cutset Regularity

In Abrahamson and Fellows [3], a graph-theoretic analog of the Myhill–Nerode theorem from formal language theory is introduced providing a useful characterization of graph families

easy to recognize for input restricted to graphs of bounded width. *Cutset regularity* gives a direct necessary and sufficient condition for the applicability of some standard algorithmic design techniques for bounded width graphs based on "finite-state" dynamic programming and related ideas.

The basic idea is as follows. If a graph family $\mathcal{F}$ is known to be cutset regular, then there exists a leaf-to-root finite-state tree automaton (linear automaton in the case of pathwidth) that recognizes $\mathcal{F}$ restricted to some treewidth (or pathwidth) bound t. Each graph can be structurally parsed; that is, we can represent the graph as a rooted tree (or rooted path) consisting of symbols taken from a finite alphabet. The symbols correspond to operations for building this graph. In turn, the structural parse is given as input to the appropriate finite-state automaton and, in linear time, membership in $\mathcal{F}$ is determined. Of course, as is the case with Courcelle's theorem (see Section 2.3.2), finding the appropriate automaton is entirely another matter. Fortunately, like the Myhill-Nerode theorem, *cutset regularity* provides an *implicit* description which can be deduced using dynamic programming and related techniques.

A proof of non-cutset regularity for $\mathcal{F}$ excludes the possible adaptation of this technique.

Cutset regularity can be considered a measure of the information flow across a bounded size cutset needed to recognize a family. The principle agents in this study are boundaried graphs, "gluing" operators, a universe called U^t_{small} and the notion of equivalence between pairs of graphs.

Definition 16 (Boundaried Graph) A *t-boundaried graph* $G = (V, E, B, f)$ is an ordinary graph $G = (V, E)$ equipped with a distinguished subset $B \subseteq V$ of t vertices, termed the *boundary* of G, together with a 1:1 map $f : B \to \{1, ..., t\}$ that *labels* the boundary.

Definition 17 (The $\oplus$ Operator) The binary operator $\oplus$ on two t-boundaried graphs is defined: $G \oplus H$ is the t-boundaried graph obtained by identifying the i^{th} boundary node of G with the i^{th} boundary node of H, for $i = 1, \ldots, t$.

Definition 18 (The $\otimes$ Operator) An *n-ary t-boundaried composition operator* $\otimes$ is defined by the data (1) a t-boundaried graph $T_\otimes = (V_\otimes, E_\otimes, B_\otimes, f_\otimes)$, (2) injective maps $f_i : \{1, \ldots, t\} \to V_\otimes$ for $i = 1, \ldots, n$. For the binary case, if G_i for $i = 1, 2$ is a pair of t-boundaried graphs $G_i = (V_i, E_i, B_i, f_i)$ then $G_1 \otimes G_2$ is defined to be the t-boundaried graph for which the ordinary underlying graph is formed from the disjoint union of G_1, G_2

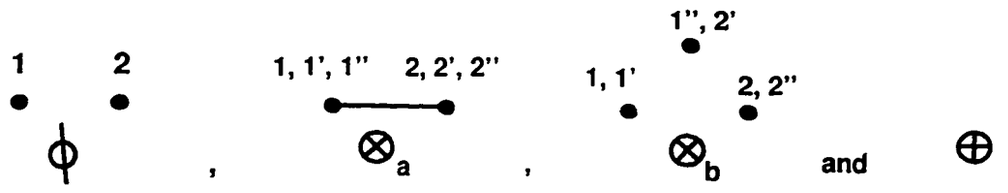

Figure 2.4: Operators for parsing treewidth 2 graphs. The first argument boundary of an operator graph is labelled $\{1', 2'\}$, the second argument boundary is labelled $\{1'', 2''\}$, and the boundary of the resulting graph is labelled $\{1, 2\}$.

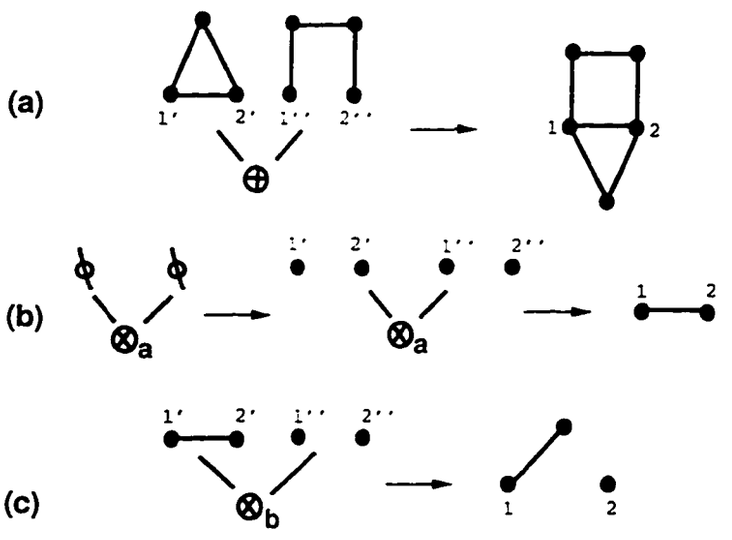

Figure 2.5: Part (a) of the diagram contains an example of the $\oplus$ operator, part (b) shows an example of the $\emptyset$ and $\otimes_a$ operators and (c) shows an example of the $\otimes_b$ operator.

and $T_\otimes$ by identifying each vertex u of B_i (for $i = 1, 2$) with its image $f_i(u)$ in $V_\otimes$. The boundary set and the labeling for $G_1 \otimes G_2$ is given by $B_\otimes$ and $f_\otimes$.

Definition 19 (Small Universe) The *small universe* U^t_{small} is the set of all t-boundaried graphs that arise in the parsing of graphs of treewidth (pathwidth) at most t.

Graphs of pathwidth at most t can be parsed using a small number of t-ary operators of boundary size t; they can also be parsed with binary operators of size $t + 1$.

Figures 2.4, 2.5 and 2.6 provide concrete examples of structural parses of graphs.

Definition 20 (Equivalence) Two t-boundaried graphs X and Y are *F-equivalent* ($X \sim_F$

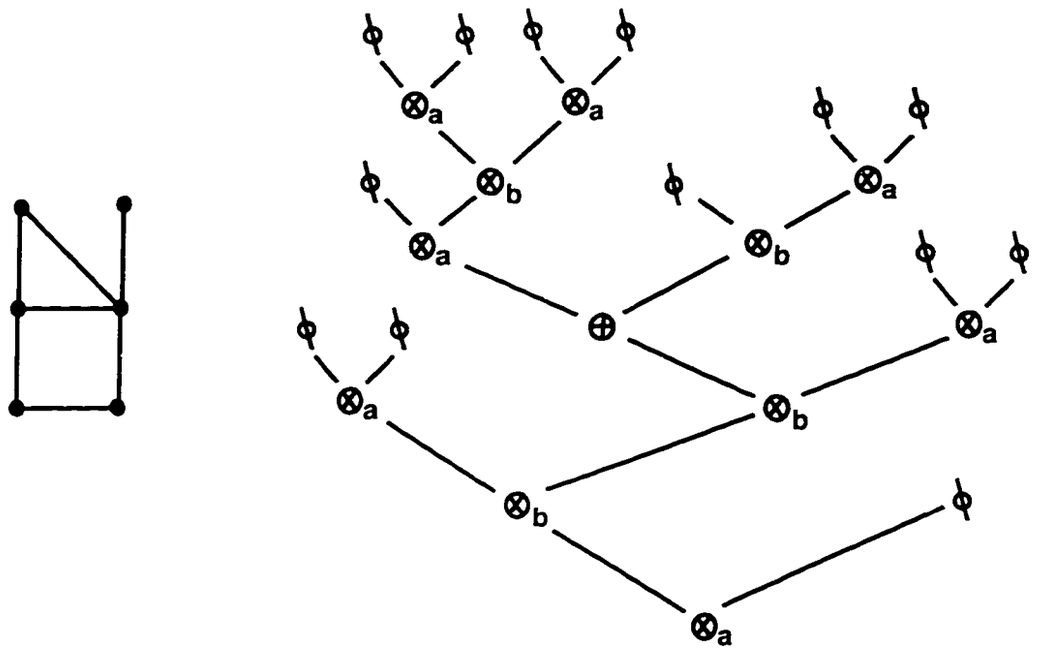

Figure 2.6: A graph with treewidth 2 and one parsing of this graph.

Y) if and only if for every t-boundaried graph Z, $X \oplus Z \in F \leftrightarrow Y \oplus Z \in F$.

The following captures the notion of "information flow" across a bounded size cutset.

Definition 21 (Cutset Regularity) Let t be a positive integer. A graph family F is *t-cutset regular* if and only if $\sim_F$ has finite index on U^t_{small}. A graph family F is *t-finite-state* if and only if there is a finite-state tree automata (linear automata in the case of pathwidth) that accepts a parse T (of a graph G of treewidth at most t) if and only if $G \in \mathcal{F}$.

Theorem 8 ([3]) A graph family F is t-finite state if and only if F is t-cutset regular. ∎

To show a family is not cutset regular, we need only exhibit an infinite set of boundaried graphs S and for each pair of boundaried graphs $X_i, X_j \in S$, a witness $Z_{i,j}$ such that $X_i \oplus Z_{i,j} \in \mathcal{F}$ and $X_j \oplus Z_{i,j} \notin \mathcal{F}$.

Chapter 3

Intervalizing Colored Graphs (DNA Physical Mapping)

This is joint work with M. R. Fellows and H. T. Wareham. The results presented here appeared in [49].

The human genome consists of 46 strands of DNA molecules each composed of approximately 130 million base pairs linearly arranged between two sugar-phosphate backbones. Each of these 46 contiguous sequences, stretching on the order of a few centimeters, wind around proteins to form a chromosome. The densely packed chromosome itself is extremely small measuring on order of only a few micrometers (during metaphase). In order to study genes and other interesting regions of the genome at the molecular level, it is necessary to unwind and decoat the DNA (called denaturing) and fragment it into much smaller and more manageable pieces. Once fragmented, we may copy (clone) the fragments many times over and use them for many different goals. For example, we may wish to determine restriction-enzyme cutting sites, the sequence of nucleotide bases in a particular fragment, or identify polymorphic DNA markers. Of course in the process of fragmentation, the position of each fragment in the genome is lost. We are left with the following fundamental goal of the PHYSICAL MAPPING problem

SEQUENCE RECONSTRUCTION

Given a set of fragments from a sequence X and a measure of overlap between pairs of sequence fragments from this set, reconstruct the order of these fragments in X.

This chapter discusses DNA physical mapping using *restriction enzyme* based techniques which we model by a problem called INTERVALIZING COLORED GRAPHS. We discuss the computational realism of this model and perform an integrated computational complexity analysis from several frameworks in order to elucidate the inherent hardness of this problem. Furthermore, we discuss known results and possible new directions.

3.1 The Model

SEQUENCE RECONSTRUCTION is typically broken into four steps (see [5, 10, 33, 47, 64]).

Step 1. Fragment the sequence X. (This step may be performed on several identical copies of X.)

Step 2. Determine a set of characteristics for each fragment.

Step 3. Determine *overlaps* between pairs of fragments.

Step 4. Using the overlap information, order the fragments into islands of contiguous fragments, termed *contigs*.

How each of the four steps is actually carried out in practice is largely a function of the type of sequence under investigation [47].

We call the fragmentation of a copy of X in Step 1 a *digest*. Where X is a piece of DNA, the fragments of X can be replicated in large quantities and the resulting copies of each fragment are termed *clones*. Typically, Step 1 is performed some number of times, say k. This value is chosen in order to ensure that, with high probability, Step 4 concludes with one contiguous fragment (contig) stretching the entire length of X. During sequencing of the yeast genome, the redundancy factor k was 8 (p. 119, [33]). Our model is predicated upon the following simple fact.

Observation 1: If two clones from Step 1 originate from the same copy of X, then they do not overlap.

Our combinatorial model is built as follows.

For each of the k iterations of Step 1, we create one vertex for each clone. If the clone was created during the i-th iteration, we color the corresponding vertex i. At the end of this phase all vertices will be assigned a color from $\{1 \ldots k\}$ with the property that all vertices corresponding to clones from the same copy of X have the same color. Two vertices are

adjacent if the corresponding clones are determined to overlap in Steps 2 and 3. The goal of the SEQUENCE RECONSTRUCTION problem is to predict additional overlap between clones from different copies of X and ultimately reconstruct the sequence X. In terms of our model, this translates to adding additional edges (between differently colored vertices) to the graph until it can be "laid out" in a linear fashion. Such a graph is termed an *interval* graph. (See [56] for related definitions and results.)

Definition 22 (Interval Graphs) An undirected graph $G = (V, E)$ is called an *interval graph* if there exists an injection $\Phi : V \to \mathcal{I} : u \mapsto I_u$ where $\mathcal{I}$ is a set of intervals of a linearly ordered set (such as the real number line) such that

$$(u, v) \in E_G \Leftrightarrow |I_u \cap I_v| \neq 0 \qquad I_u, I_v \in \mathcal{I}$$

The set $\mathcal{I}$ is called an *interval representation* of G.

Definition 23 (Vertex Colored Graphs) A graph $G = (V, E)$ with a coloring $c : V \to C$ is *properly colored*, if there is no edge between vertices with the same color.

k–INTERVALIZING COLORED GRAPHS (ICG)
Instance: A graph $G = (V, E)$ and a coloring $c : V \to \{1 : k\}$.
Parameter: k.
Question: Is there a supergraph $G' = (V, E')$ of G which is properly colored by c and which has an interval representation $\mathcal{I}$? We call G' a c–intervalization of G.

We write simply INTERVALIZING COLORED GRAPHS when referring to the non-parameterized variant of this problem.

We duly note that our model has some serious shortcomings; in particular, it is not *robust* in the (inevitable) presence of errors. The SEQUENCE RECONSTRUCTION problem is prone to four sorts of incorrect data (see [71]).

Error type 1. False negatives. These occur when the overlapping of two fragments is not detected and correspond to omissions from the edge set E of the constructed graph G. The number of such errors is largely a function of which methods are used in Steps 3 and 4 above. Detecting overlap via *restriction-fragment fingerprinting* is inherently probabilistic. Consider two clones C_1 and C_2 and let $\{c_1^1, c_2^1, \ldots, c_m^1\}$ and $\{c_1^2, c_2^2, \ldots, c_m^2\}$ be the sub-fragments obtained from the application of a *restriction enzyme* to each. (A *restriction*

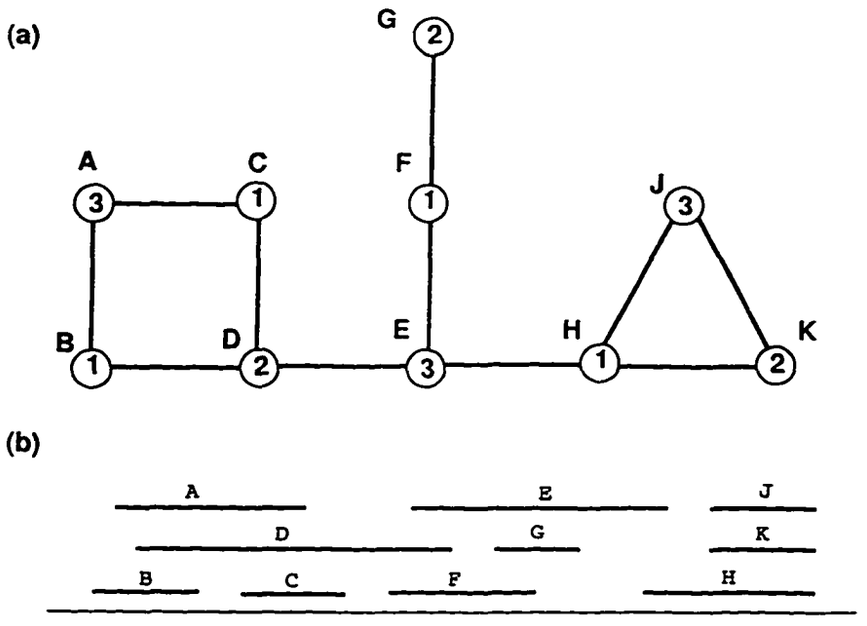

Figure 3.1: (a) A vertex colored graph G. (b) A c-intervalization of G.

enzyme separates a nucleotide sequence at well-defined sites along the strand. For example, the restriction enzyme called *EcoRI* cuts at every occurrence of the six base pair sequence GAATTC.) Through gel electrophoresis, the length of the resulting subfragments can be determined providing a *fingerprint* for C_1 and C_2. The underlying assumption is that, with high probability, two subfragments of the same length originate from the same fragment of X. These two clones are determined to overlap if there is a ***sufficiently large*** number of subfragments from C_1 with the same length as subfragments from C_2. ***Sufficiently large*** in this context means that the statistical likelihood that these two clones overlap ***in vivo*** has been exceeded. This likelihood calculation takes into consideration a number of factors including the distribution of cleavage sites in the genome, the errors in measuring subfragment lengths, and the different ways in which clones may overlap (page 117, [33] and [47]). Besides *restriction-fragment fingerprinting* there are more reliable methods of detecting overlap such as ***hybridization*** techniques (page 112, [33], [47]). Our modelling with ICG is sufficiently robust as it allows the addition of edges to G for these types of errors.

Error type 2. False positives. Of course, knowing that the fragments of two clones share one or more restriction-fragment lengths does not provide unambiguous evidence that these fragments actually are copies from the exact same region from fragment X. It is possible that clones, which in actual fact arise from separate sections of X, are predicted to overlap.

Furthermore, if the underlying nucleotide sequence of X is highly repetitive then many false positives may occur [47]. These correspond to edges erroneously included in the edge set E of G. Our model provides no manner in which to deal with such errors.

Error type 3. Chimeras. Large scale clone creation (amplification of the subfragments), may be accomplished by at least two techniques: molecular cloning and the polymerase chain reaction (PCR). We note that PCR techniques can only be applied to relatively short segments (usually a few hundred base pairs long). With either technique, the possibility of the creation of clones existing which contain two segments of a genome that are noncontiguous in vivo is introduced. These recombinant clones are termed *chimeras*. Molecular cloning uses the mechanisms of a host organism (such as *E. Coli*) to replicate the DNA. In order for the host organism to replicate this sequence, the DNA must be combined with a cloning *vector*, a DNA molecule that the host will replicate. Chimeras are created when, for example, more than one fragment is inserted into the cell of the host and correspond to erroneous vertices in the vertex set V of the graph G. Our model is not sufficiently robust to deal with such errors.

Error type 4. Lost clones. For several reasons during this four step process, fragments (and clones) may be lost. Small fragments tend to disappear during the different phases and, if a technique such as gel electrophoresis is used in Step 3, very large fragments may not behave as desired. Both pathologies result in lost clones corresponding to erroneously forgotten vertices in the vertex set V of the graph G. Our model provides no means of dealing with such errors.

Although our model provides no direct means of compensating for error types 2 through 4, it is possible to reduce the number of such errors by increasing the amount of "wet lab" (biological experimentation) effort. Repeating the mapping procedure several times for the same fragment allows the data to be cross-referenced and the number of false positives to be reduced. Error type 4 does not typically cause serious problems in mapping efforts since, as long as the redundancy factor k is sufficiently large, there will be clones of sufficiently large length arising from the same physical region of the DNA as the small lost clones.

3.2 ICG has bounded pathwidth

We begin our analysis with the following simple lemma. It provides a promising start as many problems with bounded pathwidth are easy.

Lemma 1: Let $G = (V, E)$ be a graph with a vertex coloring $c : V \to \{1 : k\}$, that is a subgraph of a properly colored interval graph G'. Then the pathwidth of G is at most $k - 1$.

Proof: The pathwidth of an interval graph is one less than its maximum clique size, which equals its chromatic number since interval graphs are perfect, see [56]. Hence the pathwidth of G' is at most $k - 1$ implying the pathwidth of G is at most $k - 1$. ∎

For instance, for bounded k, a proof that ICG is expressible in second order monadic logic or a proof that ICG is cutset regular would imply the existence of linear time algorithms to recognize the family of "yes" instances. See Section 2.3 for descriptions of these techniques. Unfortunately, the next section shows that neither of these methods are applicable except for a very limited case.

3.3 ICG is not finite state

Let t be the size of the boundary. We remind the reader that the method of argument is much the same as any standard application of the Myhill–Nerode theorem to show non-regularity for a formal language. It is sufficient to exhibit an infinite set of boundaried graphs S and for each pair $x_i, x_j \in S$, a witness $z_{i,j}$ such that $x_i \oplus z_{i,j} \in S$ and $x_j \oplus z_{i,j} \notin S$. See Section 2.3.3 for a more complete explanation of the theory.

Definition 24 (Properly Less Intervals) We say that an interval $I_v < x$ (I_v *is properly less than* x) if and only if for all y in I_v, $y < x$. Furthermore $I_v < I_w$ (I_v *is properly less than* I_w) if and only if for all x in I_v and all y in I_w, $x < y$. If an interval I_v is a proper subinterval of I_w, then we say that I_v *lands on* I_w.

Theorem 9: k-ICG is not t-finite-state for all $k \geq 3$ and $t \geq 1$.

Proof: We argue using 3-colored graphs and $t = 1$. For $i \geq 1$, let $X_i = (V, E, B, f)$ be constructed as follows:
$V = \{a_0, a_1, a_2, a_3, a_4\} \cup \{b_0, b_1, \ldots, b_{2i}\}$;
$E = \{(a_0, a_1), (a_0, a_2), (a_1, a_2), (a_3, a_4), (b_0, a_1), (b_0, a_2), (b_{2i}, a_3), (b_{2i}, a_4)\} \cup \{(b_k, b_{k+1}) | 0 \leq k < 2i\}$;
$B = \{b_0\}$; $f(b_0) = 1$.
Let $c : V \to \{1, 2, 3\}$ be the coloring of X_i: $c(a_0) = 1$, $c(a_1) = c(a_3) = 2$, $c(a_2) = c(a_4) = 3$, $c(b_k) = (k \mod 2) + 1$, for $0 \leq k \leq 2i$.

For $j \geq 1$, let $Y_j = (V', E', B', f')$ be constructed as follows:
$V' = \{h_0, h_1, \ldots, h_{4j}\}$;
$E' = \{(h_k, h_{k+1}) | 0 \leq k < 4j\}$; $B' = \{h_0\}$; $f'(h_0) = 1$.
Let $c' : V' \rightarrow \{1, 2, 3\}$ be the coloring of Y_j: $c'(h_0) = 1$, $c'(h_k) = 3$ if $k = 1 \pmod 2$. $c'(h_k) = 1$ if $k = 2 \pmod 4$, $c'(h_k) = 2$ if $k = 0 \pmod 4$.

Claim 1.1: $X_i \ominus Y_i$ is c–intervalizable.

It is easy to verify that the following is indeed a c–intervalization of G (See Figure 3.2). For convenience, we use the real number line. Let $\epsilon = \frac{1}{3}$.

Map I_{a_0} to the interval $[-\epsilon..0]$.
Map I_{a_1} and I_{a_2} to $[-\epsilon..\epsilon]$.
Map $I_{b_0 = h_0}$ to $(0..2 + \epsilon]$.
Map I_{h_k} to $[2k - \epsilon..2k + 2 + \epsilon]$, $0 < k \leq 2i$.
Map I_{a_3} and I_{a_4} to $[4i + 2..4i + 2 + \epsilon]$.
Map I_{h_k} to $[k + 1 - \epsilon..k + 1 + \epsilon]$, for even $k > 0$.
Map I_{h_k} to $[k + \frac{\epsilon}{2}..k + 2 - \frac{\epsilon}{2}]$, for odd k.

Claim 1.2: For $i < j$, $X_i \oplus Y_j$ is not c–intervalizable.

W.l.o.g. assume $I_{a_1} < I_{a_3}$.

Let $\alpha = max(\{x | x \in I_{a_0} \cap I_{a_1} \cap I_{a_2}\})$, $\beta = max(\{x | x \in I_{a_1} \cap I_{a_2} \cap I_{b_0}\})$ and $\omega = min(\{x | x \in I_{a_3} \cap I_{a_4} \cap I_{b_{2i}}\})$. Clearly, $\alpha < I_{b_k} < \omega$, for $0 \leq k < 2i$, since every color is present at both points and each b_k is on a path from a_0 to a_3.

Clearly, $\alpha < I_{h_k} < \omega$, for $1 \leq k \leq 4j$, since every color is present at both of these points and there exists a path from each h_k to a_0 and a_3.

In fact $\beta < I_{h_k}$ for $k > 0$. Suppose otherwise; that is, $\alpha < I_{h_k} \leq \beta$. Then it is must be the case that $\alpha < I_{h_k} \leq \beta$ for all $k > 0$. But colors 2 and 3 are present at every point in this interval. This contradicts the proper coloring.

In any c–intervalization, there exists an interval strictly between I_{a_0} and I_{b_0} where the color 1 is not present. We call this interval ι_0. The two colored path $b_0 b_1 \ldots b_{2i} a_2$ creates i intervals strictly between $I_{b_{k-1}}$ and $I_{b_{k+1}}$ for odd k, $1 \leq k \leq 2i - 1$, where the color 1 is not present. We refer to these intervals as $\iota_1, \iota_2, \ldots, \iota_i$.

There are j vertices in the subgraph Y_j with color 1: $h_2, h_6, \ldots, h_{4j-2}$. In any c–intervalization, the intervals corresponding to these vertices must land in ι_k for some k, $0 \leq k \leq i$. Note,

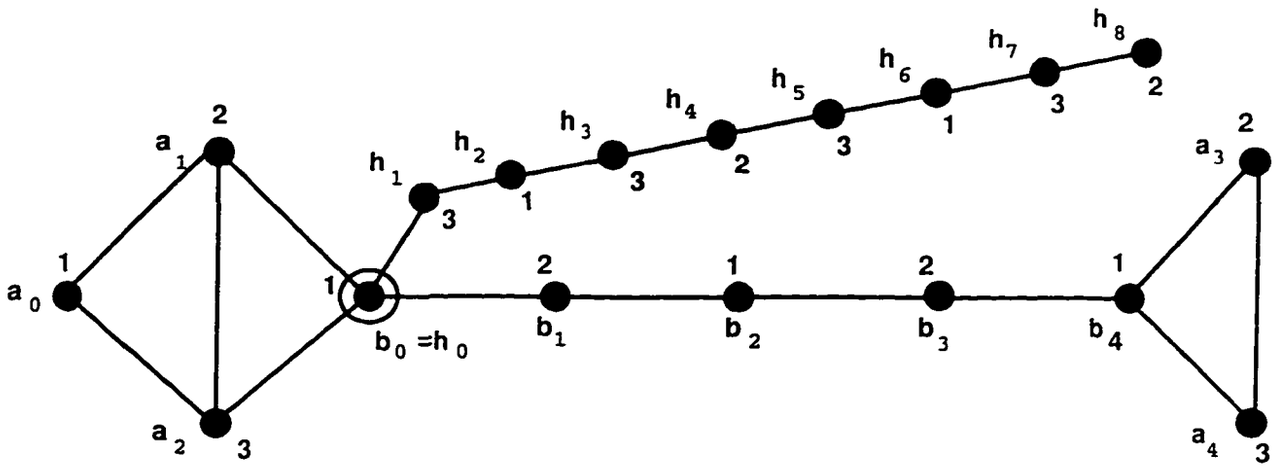

Figure 3.2: An example "yes" instance: $X_2 \in Y_2$.

however, that $i_0 < 3$ implying there exists in fact only i intervals for these j vertices. Therefore, there exists an ι_r such that $I_{h_m} \subset \iota_r$ and $I_{h_{m'}} \subset \iota_r$, $m \neq m'$.

W.l.o.g. assume $I_{h_m} < I_{h_{m'}}$. Consider the path $h_m h_{m+1} h_{m+2} \cdots h_{m'-2} h_{m'-1} h_{m'}$ in Y_j. Now $c'(h_{m+1}) = c'(h_{m'-1}) = 3$. Let $\gamma = max(\{x | x \in I_{h_m} \cap I_{h_{m+1}} \cap \iota_r\})$ and $\delta = min(\{x | x \in I_{h_{m'}} \cap I_{h_{m'-1}} \cap \iota_r\})$. Since $\iota_r \subset I_{b_k}$ for some k, every color is present at both points implying the intervals of all vertices in the subpath $h_{m+2} h_{m+3} \ldots h_{m'-2}$ must lie strictly between γ and δ. It is easy to verify from the construction that any such subpath necessarily has a vertex colored 2. As $[\gamma..\delta] \subset \iota_r$, color 2 is present at every point in this interval and the placement of the intervals for such vertices contradicts the proper coloring. ∎

For the remaining cases, we have the following "positive" theorem.

Theorem 10: For all t and for $k \leq 2$, k-ICG is t-finite-state.

Proof: This proceeds in two steps. Firstly, we provide an alternate characterization of the family of 2-colored caterpillars and secondly, express this characterization in the second order monadic logic of graphs.

Definition 25 (Caterpillar Graphs) A graph $G = (V, E)$ is a *caterpillar* if there exists a path P in G such that for all $v \in V_G - V_P$, there exists a unique vertex $u \in V_P$ such that $(u, v) \in E_G$.

Claim 1.3: A 2-colored graph G is c-intervalizable if and only if G is a properly colored caterpillar.

($\Leftarrow$) Assume G is connected. If it is not, repeat the following argument for each connected component. If G is a caterpillar, then by definition, there exists a path P in G such that for all $v \in V_G - V_P$, there exists a unique vertex $u \in V_P$ such that $(u, v) \in E_G$. Let $\mathcal{I}$ be a c-intervalization of P. Let P be written as $v_1 v_2 \ldots v_m$ where $(v_i, v_{i+1}) \in E_P$ for $1 \leq i < m \leq n$. Consider some $v \in V_G - V_P$ and let u be the unique vertex in V_P such that $(u, v) \in E_G$. If $v = v_1$ or $v = v_m$ it is easy to see how the c-intervalization can be extended. Otherwise, if $v = v_i$, then there exists an interval between $I_{v_{i-1}}$ and $I_{v_{i+1}}$ since $c(v_{i-1}) = c(v_{i+1})$. Place I_v in this interval.

($\Rightarrow$) Again assume G is connected. Let G' be a c-intervalization of G, a 2-colored graph. Let P be the graph induced by the following set of vertices: $V_P = \{v | I_v \not\subset I_w, \forall w \in V_G, w \neq v\}$. It is easy to see that P must in fact be a path since G is connected and properly 2 colored. Now consider a vertex $u \in V_G - V_P$. Since $I_u \subset I_w$ for some $w \in V_G$, u is adjacent to only w in G'. Otherwise, suppose u is adjacent to some vertex x, $x \neq w$. Then x is adjacent to w implying the existence of a 3 cycle in a properly 2-colored graph. This contradicts the proper 2 coloring. Hence, there exists a path in G such that every vertex is either on this path or adjacent to a unique vertex of the path and, by definition, G is a caterpillar.

We express the family of caterpillar graphs in the language of second monadic logic for graphs (see Section 2.3.2). We may assume the G is connected since we can identify the connected components of G in linear time.

$$
\begin{aligned}
\text{colored-caterpillar}(G) \quad &\leftrightarrow \quad \exists V_1 \subseteq V : \exists V_2 \subseteq V : \exists V_3 \subseteq V : \exists V_4 \subseteq V : \text{partition}(V_1, V_2) \wedge \\
&\quad \text{partition}(V_3, V_4) \wedge \text{proper-coloring}(V_1, V_2) \wedge \text{cater-legs}(V_3, V_4) \wedge \\
&\quad \text{cater-body}(V_4) \\
\text{partition}(V', V'') \quad &\leftrightarrow \quad \forall v \in V : (v \in V' \vee v \in V'') \wedge \neg(v \in V' \wedge v \in V'') \\
\text{proper-coloring}(V', V'') \quad &\leftrightarrow \quad \forall u \in V : \forall v \in V : adj(u, v) \rightarrow ((u \in V' \wedge v \in V'') \vee \\
&\quad (u \in V'' \wedge v \in V')) \\
\text{cater-legs}(V', V'') \quad &\leftrightarrow \quad (\forall u \in V' : \text{degree-one}(u, V' \vee V'')) \\
&\quad \wedge (\forall u \in V : u \in V' \rightarrow \exists v \in V'' : adj(u, v)) \\
\text{cater-body}(V'') \quad &\leftrightarrow \quad ((\exists u \in V'' : \exists v \in V'' : u \neq v \wedge \text{degree-one}(u, V'') \wedge \text{degree-one}(v, V'')) \wedge
\end{aligned}
$$

$$(\forall u \in V'' : \neg\text{degree-one}(u, V'') \rightarrow \text{degree-two}(u, V''))) \vee$$
$$(\exists v \in V'' : \forall w \in V'' : w = v)$$

$$\text{degree-one}(u, V') \leftrightarrow \exists v \in V' : adj(u, v) \wedge (\forall w \in V' : (w \neq v \wedge w \neq u) \rightarrow \neg adj(u, w))$$

$$\text{degree-two}(u, V') \leftrightarrow \exists v \in V' : \exists w \in V' : v \neq w \wedge u \neq v \wedge u \neq w \wedge adj(u, v) \wedge adj(u, w) \wedge$$
$$(\forall z \in V' : z \neq u \wedge z \neq v \wedge z \neq w \rightarrow \neg adj(u, z))$$

By the results of [34] the problem is finite-state for all t. ∎

See also [22] for an alternative proof that ICG restricted to 2 colors has a linear time algorithm.

3.4 Hardness for ICG

Although the preceding section provides some "bad news" for k–ICG, it still may be the case that it is FPT via some other algorithmic route. Unfortunately, the next result blackens the picture considerably showing no such method will work unless very unlikely computational complexity collapses occur. By a rather complicated parameterized reduction from the problem k–INDEPENDENT SET, we show the following.

Theorem 11: k–ICG is $W[1]$–hard.

The theorem follows from the fact the k–INDEPENDENT SET is complete for $W[1]$ (see [41]) and Claim 1.10.

k–INDEPENDENT SET
Instance: A graph $G = (V, E)$, an integer k.
Parameter: k.
Question: Does there exist a set $V' \subseteq V$, $|V'| \geq k$, such that, for all $u, v \in V'$, $(u, v) \notin E$?

Let $G_{IS} = (V, E)$ be a graph with the vertices labeled from $1 \ldots |V| = n$. Let $m = n\binom{n}{2}$ and $m' = n\binom{n}{2} - n + 1$. From G_{IS} we create a graph $G_{ICG} = (V', E')$ that has coloring $c : V' \rightarrow \{1, \ldots, 3k, block, floor, queen, slide\}$. G_{ICG} has four sets of components.

Component 1: Vertex Selection Components (B's and Queen B's) We create k decision components $\mathcal{B}_i$ each of which consists of m' B components where: $V(B) = \{b_1, \ldots, b_5\}$ and $E(B) = \{(b_j, b_{j+1}) \mid 1 \leq j < 5\}$.

Let $B' = B_1 \cup B_2 \cup \ldots \cup B_{m'}$. Within B', identify vertex $b_5 \in B_j$ with vertex $b_1 \in B_{j+1}$ for $1 \leq j < m'$.

Let $\mathcal{B}_i = B' \cup \{\hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{\binom{n}{2}}\} \cup \{\underline{b}_1, \underline{b}_2\}$. We add the following edges to $\mathcal{B}_i$: $\{(\hat{b}_j, b) | 1 \leq j \leq \binom{n}{2}, \forall b \in B_{n(j-1)+1}\}$ and $\{(\underline{b}_1, b_1), (\underline{b}_2, b_5) \mid b_1 \in B_1 \text{ of } \mathcal{B}_i, b_5 \in B_{m'} \text{ of } \mathcal{B}_i\}$.

Let $QB = \{B_{n(j-1)+1} \cup \hat{b}_j | B_{n(j-1)+1} \in \mathcal{B}_i, \ 1 \leq i \leq k, \ 1 \leq j \leq \binom{n}{2}\}$ (the Queen B's).

We color each $\mathcal{B}_i$ as follows: $c(b_1) = c(b_5) = 3(i-1)+1$; $c(b_2) = c(b_4) = 3(i-1)+2$; $c(\underline{b}_1) = c(\underline{b}_2) = 3(i-1)+3$; $c(b_3) = floor$; $c(\hat{b}_j) = queen$ for all j.

Component 2: Flowers and Flower Pots Our second component $\mathcal{F}$ consists of m F components (flowers) where $V(F) = Q \cup R \cup S \cup T$ where $Q = \{q_j | 1 \leq j \leq k+1\}$, $R = \{r_j | 1 \leq j \leq k\}$, $S = \{s_j | 1 \leq j \leq k+1\}$, and $T = \{t_1, t_2\}$.

$E(F)$ contains all the edges necessary so that (1) the vertices of Q form a clique, (2) the vertices of R form a clique, (3) the vertices of S form a clique, (4) t_1 is adjacent to every vertex in Q and R, and (5) t_2 is adjacent to every vertex in R and S.

Each vertex of $Q - \{q_{k+1}\}$ and $S - \{s_{k+1}\}$ is assigned a unique color from $\{3j+2 \mid 0 \leq j \leq k-1\}$ and each vertex of R is assigned a unique color from $\{3j+1 \mid 0 \leq j \leq k-1\}$. Additionally, let $c(t_1) = c(t_2) = floor$, $c(q_{k+1}) = c(s_{k+1}) = block$.

Let $\mathcal{F} = F_1 \cup F_2 \cup \ldots \cup F_m$ identifying all the vertices in the set S of F_j with their like-colored vertices in the set Q of F_{j+1}, $1 \leq j < m$.

For each decision component $\mathcal{B}_i$, we add edges between $\underline{b}_1$ and every vertex of Q in F_1 and edges between $\underline{b}_2$ and every vertex of S in F_m of $\mathcal{F}$.

Lastly, let $\hat{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_{\binom{n}{2}-1}\}$. We add edges from $\hat{f}_i$ to every vertex in the set S and t_2 of F_{ni}, and from $\hat{f}_i$ to t_1 of F_{ni+1}. Let $c(\hat{f}_i) = slide$.

Note that the number of flowers in this component is equal to the total possible number of edges in graph G_{IS} multipled by n. Establish a bijection between groups of n contiguous F components (flower pots) and all possible edges in G_{IS}. For instance, let us use the bijection in which the pair $\{1,2\}$ corresponds to $\{F_1, \ldots, F_n\} \subseteq \mathcal{F}$, $\{1,3\}$ corresponds to $\{F_{n+1}, \ldots, F_{2n}\}$ and $\{n-1, n\}$ corresponds to $\{F_{n\binom{n}{2}-n}, \ldots, F_m\}$. This bijection will be used in component 4 described below.

Component 3: Rocks Let C_1 and C_2 be two cliques of size $3k+4$. Every vertex in C_1 (and C_2) is assigned a unique color. We identify the vertices of C_1 which have been assigned colors from the set $\{3i+2 \mid 0 \leq i \leq k-1\} \cup \{block\}$ with their like-colored vertices in the set Q of F_1. Similarly, we identify the vertices of C_2 with their like-colored vertices in the set S of F_m.

Component 4: Edge-Constraint Components (Enforcers) Use the bijection described above for the $\mathcal{F}$ component, and let $F_{p+1}, \ldots, F_{p+n}$ be the corresponding F components for a pair of vertices $\{u, v\}$. If $(u, v) \in E(G_{IS})$ then, we create a component N where $V(N) = \{z_1, \ldots, z_5\}$ and $E(N) = \{(z_i, z_j) \mid 1 \leq i \neq j \leq 4\} \cup \{(z_i, z_j) \mid 2 \leq i \neq j \leq 5\}$.

Additionally, we place edges between z_1 and every vertex of R in F_u and we place edges between z_5 and every vertex of R in F_v. For each F_i, $i \neq u$ or v, add a vertex d_i s.t. d_i is adjacent to t_1 and t_2 of F_i. If $(u, v) \notin E(G_{IS})$ then we add such a vertex d to every F component (in the flower pot).

We color these components as follows: $c(z_1) = c(z_5) = slide$; $c(z_2) = queen$; $c(z_3) = floor$; $c(z_4) = c(d_i) = block$ for all i.

Figure 3.3 contains a sample construction of a graph G_{ICG} for a given graph G_{IS}.

Let $\alpha = max(\{x \mid x \in \bigcap_{y \in C_1} I_y\})$ and $\omega = min(\{x \mid x \in \bigcap_{y \in C_2} I_y\})$.

Claim 1.4: If $\mathcal{I}$ is a c-intervalization of $G_{ICG} = (V, E)$ using at most $3k+4$ colors, then for all $v \in V - (C_1 \cup C_2)$ either $\alpha < I_v < \omega$ or $\alpha > I_v > \omega$.

Proof: Let $\mathcal{I}$ be a c-intervalization of G_{ICG}. Clearly $\alpha \neq \omega$. Consider the first case where $\alpha < \omega$ and let v be any vertex in $V - (C_1 \cup C_2)$. Let $P = v_0 v_1 \ldots v_{i-1} v_i = v\, v_{i+1} \ldots v_l$ such that $v_0 \in C_1$, $v_l \in C_2$ and $v_1, \ldots, v_{l-1} \in V - (C_1 \cup C_2)$. Suppose $\exists x \in I_v$ s.t. $x \leq \alpha$ (or suppose $x \geq \omega$). The intervals corresponding to vertices $v_i, \ldots v_l$ ($v_0, \ldots, v_i$) form a non-disjoint chain and $\omega \in I_{v_l}$ ($\alpha \in I_{v_0}$). Therefore $\alpha \in I_z$ ($\omega \in I_z$) for some $z \in P - \{v_0, v_l\}$ and since $|C_1| = 3k+4$ ($|C_2| = 3k+4$), $3k+5$ intervals intersect at α (at ω). Hence $\nexists x \leq \alpha$ ($\nexists x \geq \omega$).

A similar argument establishes the second inequality. ∎

For the remainder of this section, we assume without loss of generality that $\alpha < \omega$. Let $\beta_i = min(\{x \mid x \in \bigcap_{y \in t_1 \cup Q} I_y\})$, $\gamma_i = max(\{x \mid x \in \bigcap_{y \in t_1 \cup R} I_y\})$, $\delta_i = min(\{x \mid x \in \bigcap_{y \in t_2 \cup R} I_y\})$, $\epsilon_i = max(\{x \mid x \in \bigcap_{y \in t_2 \cup S} I_y\})$ where $t_1, t_2, Q, R, S \in F_i$.

Claim 1.5: In any c-intervalization of the graph induced by $\mathcal{F} \cup C_1 \cup C_2$ the following holds for the m F components of $\mathcal{F}$:

$$\alpha < \beta_1 \leq \gamma_1 < \delta_1 \leq \epsilon_1 < \ldots < \beta_m \leq \gamma_m < \delta_m \leq \epsilon_m < \omega.$$

Proof: We show this by induction on the size of $\mathcal{F}$. Let $\mathcal{F}$ consist of one F component. That $\alpha < \beta_1, \gamma_1, \delta_1, \epsilon_1 < \omega$ follows from Claim 1.4. Suppose $\gamma_1 < \beta_1$; that is, $max(\{x \mid x \in \bigcap_{y \in t_1 \cup R} I_y\}) < min(\{x \mid x \in \bigcap_{y \in t_1 \cup Q} I_y\})$. Clearly, it must be the case that $\gamma_1 < \{z | \forall z \in \bigcap_{y \in Q} I_y\}$ but $Q \subseteq C_1$, which implies that $\gamma_1 < \alpha$. This contradicts Claim 1.4. A symmetric argument can be given to show that $\delta_1 \not> \epsilon_1$.

Clearly $\delta_1 \neq \gamma_1$, so suppose $\delta_1 < \gamma_1$. Since $c(t_1) = c(t_2)$, δ_1 can not be in the interval $(\beta_1 \ldots . \gamma_1)$ so $\gamma_1 < \beta_1$ and $I_{t_2} < \beta_1$ implying $\epsilon_1 < \beta_1$. Note however that the vertices of Q and S are colored identically. Therefore $\epsilon_1 < \bigcap_{y \in Q} I_y$ but $Q \subseteq C_1$ implies that $\epsilon_1 < \alpha$, again contradicting Claim 1.4.

Assume that the above property holds for $\mathcal{F}$ consisting of $k < m$ components. Let $\mathcal{F}$ consist of m components. The induction hypothesis holds for the graph induced by removing the m-th flower F_m appropriately. We are required to show that $\epsilon_{m-1} < \beta_m \leq \gamma_m < \delta_m \leq \epsilon_m < \omega$ in any c-intervalization.

Clearly $\beta_m \neq \epsilon_{m-1}$ so suppose $\beta_m < \epsilon_{m-1}$. Because $t_1 \in F_m$ is adjacent to all $s \in S_{m-1} = Q_m$ it is easy to see that $\beta_m > \beta_{m-1}$. In fact, the only interval where β_m may be placed is $(\gamma_{m-1} \ldots \delta_{m-1})$. If this were the case, then $I_{t_1 \in F_m} < \delta_{m-1}$. But $t_1 \in F_m$ is adjacent to all $r \in R$ of F_m implying that $\bigcap_{r \in R_{m-1}}$ has non-empty intersection with $\bigcap_{r \in R_m}$. These two sets are colored identically contradicting the proper coloring.

Suppose $\gamma_m < \beta_m$; that is, $max(\{x \mid x \in \bigcap_{y \in t_{1,m} \cup R_m} I_y\}) < min(\{x \mid x \in \bigcap_{y \in t_{1m} \cup Q_m} I_y\})$. This implies that $max(\{x \mid x \in \bigcap_{r \in R_m} I_r\}) < min(\{x \mid x \in \bigcap_{q \in Q_m = S_{m-1}} I_q\})$. Hence $I_{t_1 \in F_m}$ spans the interval $(\gamma_m \ldots \beta_m)$. But $\epsilon_{m-1} < \beta_m$ and $t_2 \in F_{m-1}$ is adjacent to all vertices in the set S $(= Q)$ of $F_{m-1(=m)}$. Therefore there exists a point x such that $x \in I_{t_1 \in F_m}$ and $x \in I_{t_2 \in F_{m-1}}$ contradicting the proper coloring.

Clearly $\delta_m \neq \gamma_m$ so suppose $\delta_m < \gamma_m$. Now $\delta_m > \delta_{m-1}$ since the colors present at δ_{m-1} are identical to the colors present at δ_m and $t_2 \in F_m$ is adjacent to all vertices of R_m. In fact, the only valid interval for δ_m is $(\epsilon_{m-1} \ldots \beta_m)$. If this were the case, then $\epsilon_{m-1} < I_{t_2 \in F_m} < \beta_m$ implying that $\epsilon_{m-1} < \epsilon_m < \beta_m$. But this contradicts the proper coloring since the vertices

of S_m are colored identically to the vertices of $S_{m-1} = Q_m$.

Suppose $\epsilon_m < \delta_m$. Then $max(\{x \mid x \in \bigcap_{s \in S_m} I_s\}) < min(\{x \mid x \in \bigcap_{r \in R_m} I_r\})$. Since $S_m \subseteq C_2$, $\omega < min(\{x \mid x \in \bigcap_{r \in R_m} I_r\})$ which contradicts Claim 1.4. ∎

We say that a B component *lands* in the p-th F component if $\gamma_p < I_{b_i} < \delta_p$.

Claim 1.6: Consider some B_i in one of the decision components. It must be the case that $\gamma_p < I_{b_i \in B_i} < \delta_p$ in any c-intervalization of G_{ICG} for $1 \leq p \leq m$.

Proof: By Claim 1.4, $\alpha < I_{b_3} < \omega$. Since $c(b_3) = c(t_1) = c(t_2)$ it can not be the case that either $\beta_p \leq I_{b_3} \leq \gamma_p$ or $\delta_p \leq I_{b_3} \leq \epsilon_p$, $1 \leq p \leq m$.

Suppose $\epsilon_p < I_{b_3} < \beta_{p+1}$. Since b_3 is adjacent to b_2 and by the definition of ϵ and β, $\exists x \in I_{b_2} \cap I_{b_3}$ such that $x \in \bigcap_{q \in S_{p-1} = Q_p} I_q$. This contradicts the proper coloring, since for some such q, $c(q) = c(b_2)$. This same argument can be used to show that it cannot be the case that $\alpha < I_{b_3} < \beta_1$ nor $\epsilon_m < I_{b_3} < \omega$.

The remaining case is to place I_{b_3} so that it is properly greater than γ_p and properly less than δ_p. It is easy to verify that the intervals corresponding to $\{b_1, \ldots, b_5\}$ are able to find a color-respecting c-intervalization. ∎

Claim 1.7: For each decision component, $B_1, B_2 \ldots, B_{m'}$ must land in m' contiguous flowers $F_p, F_{p+1}, \ldots, F_{p+m'-1}$ and furthermore, $p \leq n$.

Proof: It is easy to show by induction on the length of the chain of B components that if $\gamma_p < I_{b'_3} < \delta_p$, then $\gamma_{p+1} < I_{b''_3} < \delta_{p+1}$, $b'_3 \in B_i$, $b''_3 \in B_{i+1}$, and $1 \leq i \leq m' - 1$. Since these $m' = n\binom{n}{2} - n + 1$ B components land in contiguous flowers and there are $m = n\binom{n}{2}$ such F components, it follows that the first B_1 of each of the k decision components $\mathcal{B}_j$ must land in one of the first n flowers. ∎

Claim 1.8: No two distinct elements of QB may land in the same flower F_p.

Proof: Recall that QB is the set of "queen" B's and that there exists a vertex $\dot{b}$, $c(\dot{b}) = queen$, such that $\dot{b}$ is adjacent to every vertex in the B component. Let $\dot{B} \in QB$. By Claim 1.6, $\gamma_p < I_{b_3} < \delta_p$. Note that $c(b_1) = c(b_5)$ and that this color is present at every point in the interval $(\gamma_p \ldots \delta_p)$. Hence $I_{b_1} < \gamma_p$ and $\delta_p < I_{b_5}$. Since $I_{\dot{b}} \cap I_{b_1} \neq \emptyset$ and $I_{\dot{b}} \cap I_{b_3} \neq \emptyset$ and $I_{\dot{b}} \cap I_{b_5} \neq \emptyset$, the color *queen* is present at every point between γ_p and δ_p. Therefore only one member of QB may land in a particular flower. ∎

Let $F_p, F_{p+1}, \ldots, F_{p+n-1}$ be the n contiguous flowers associated with an edge $(u, v) \in E(G_{IS})$. Let N be the enforcer associated with this edge. Without loss of generality we assume that $u < v$. Let $\zeta = min(\{x \mid x \in I_{z_2} \cap I_{z_3} \cap I_{z_4}\})$.

Claim 1.9: In a c-intervalization, it must be the case that either $\gamma_{p+u-1} < \zeta < \delta_{p+u-1}$ or $\gamma_{p+v-1} < \zeta < \delta_{p+v-1}$. Furthermore, if a member of QB lands in F_u, then $\gamma_{p+v-1} < \zeta < \delta_{p+v-1}$.

Proof: It is easy to verify that the only intervals where ζ could possibly reside and yet still respect the proper coloring is between γ_i and δ_i for $1 \leq i \leq m$. We first argue that $\beta_p < \zeta < \epsilon_{p+n-1}$ (enforcers stay within their flower pots). But this is clearly the case because of the placement of the $\hat{F}$ vertices where $c(\hat{F}) = \textit{slide}$. This guarantees that $I_{z_1} > \beta_p$ and $I_{z_5} < \epsilon_{p+n}$.

Now recall that for every F_x where $x \neq u$ or v, a vertex d was created and made adjacent to t_1 and t_2. Therefore, in any c-intervalization, $\exists x \in I_d$ s.t. $x \leq \gamma_x$ and $\exists x \in I_d$ such that $x \geq \delta_x$. Therefore at every point between γ_x and δ_x, $c(d) = \textit{block}$ is present implying that ζ can not lie in this interval.

To show the second statement in our claim, suppose a member of QB lands in F_u. By a similar argument to that in the proof of Claim 1.8, it is true that at every point in the interval $(\gamma_u \ldots \delta_u)$ the color *queen* is present. Since $c(z_2) = \textit{queen}$, ζ can not lie in this interval. ∎

See Figure 3.4 for a high-level description of the reduction.

Claim 1.10: G_{IS} has an independent set of size k if and only if G_{ICG} has a c-intervalization using at most $3k + 4$ colors.

Proof: Suppose G_{ICG} has a c-intervalization $\mathcal{I}$ with $3k + 4$ colors. Let $IS = \{j \mid B_i$ of B_i lands in F_j, $1 \leq i \leq k\}$. By Claims 1.7 and 1.8, $|IS| = k$ and for all $j \in IS$, $1 \leq j \leq n$. We claim that IS constitutes an independent set for G_{IS}. Consider $x \in IS$. By Claim 1.7, $F_x, F_{x+1}, \ldots, F_{m'+x-1}$ have $B_1, B_2, \ldots, B_{m'}$ land in them respectively and $F_x, F_{n+x}, F_{2n+x}, \ldots, F_{m'+x-1}$ have members of QB land in them. By Claim 1.9, the enforcer N corresponding to any edge of the form $(x, y) \in E(G_{IS})$ must land in the F component corresponding to y. Therefore $F = F_{pn+y}$, for some $p \in \{0, \ldots, \binom{n}{2} - 1\}$ must not have a member of QB land in it, since if it does N is not able to land anywhere. This implies that F_y does not have a member of QB land in it. Hence $y \notin IS$.

Suppose G_{IS} has a k element independent set and let IS be such a set. For each $x \in IS$, we let B_i of some unique B_i, land in F_x. By Claim 1.7, flowers $F_x, F_{x+1}, \ldots, F_{m'+x-1}$ have $B_1, B_2, \ldots, B_{m'}$ land in them respectively. $B_1, B_{n+1}, B_{2n+1}, \ldots, B_{m'}$ are members of QB and therefore the flowers F_x, F_{n+x}, $F_{2n+x}, \ldots$, $F_{m'+x-1}$ have members of QB land in them. Since IS constitutes an independent set, if $(x, y) \in E(G_{IS})$, then $y \notin IS$ and by a similar chain of reasoning to the above the flowers $F_y, F_{n+y}, F_{2n+y}, \ldots, F_{m'+y-1}$ do not have members of QB land in them. By Claim 1.9, the enforcer N for edge (x, y) can land in the F component corresponding to y. ∎

In fact, our reduction also shows the following about the non-parameterized version of INTERVALIZING COLORED GRAPHS.

Theorem 12 (Also see [57]) ICG is $\mathcal{NP}$-complete.

Proof: To see this, simply note that in our reduction the function $f(k)$ is polynomial. We can verify that a c-intervalization G' of G is an interval graph [69] and properly colored in polynomial time. ∎

Note that Golumbic, Kaplan and Shamir [57] proved independently the $\mathcal{NP}$-completeness for this problem. However, their reduction does not constitute a parameterized reduction.

As noted in Section 2.2.3, parameterized reductions tend to be quite technically intricate. Heuristics for designing combinatoric mechanisms (*gadgets*) are of great use and provide good starting points in the early design stages of a reduction. The analysis of ICG has yielded the following

THE ⋆ HEURISTIC

The combinatoric mechanisms (*gadgets*) underlying proofs of non-finite stateness are functionally equivalent to those mechanisms used in proofs of W-hardness.

A closely related problem to INTERVALIZING COLORED GRAPHS- TRIANGULATING COLORED GRAPHS [20]- also exhibits this behaviour (see [17] for other examples).

3.5 Further Results

The problem INTERVALIZING COLORED GRAPHS has received considerable attention over the past few years. At roughly the same time, Fellows, Hallett and Wareham [49] and

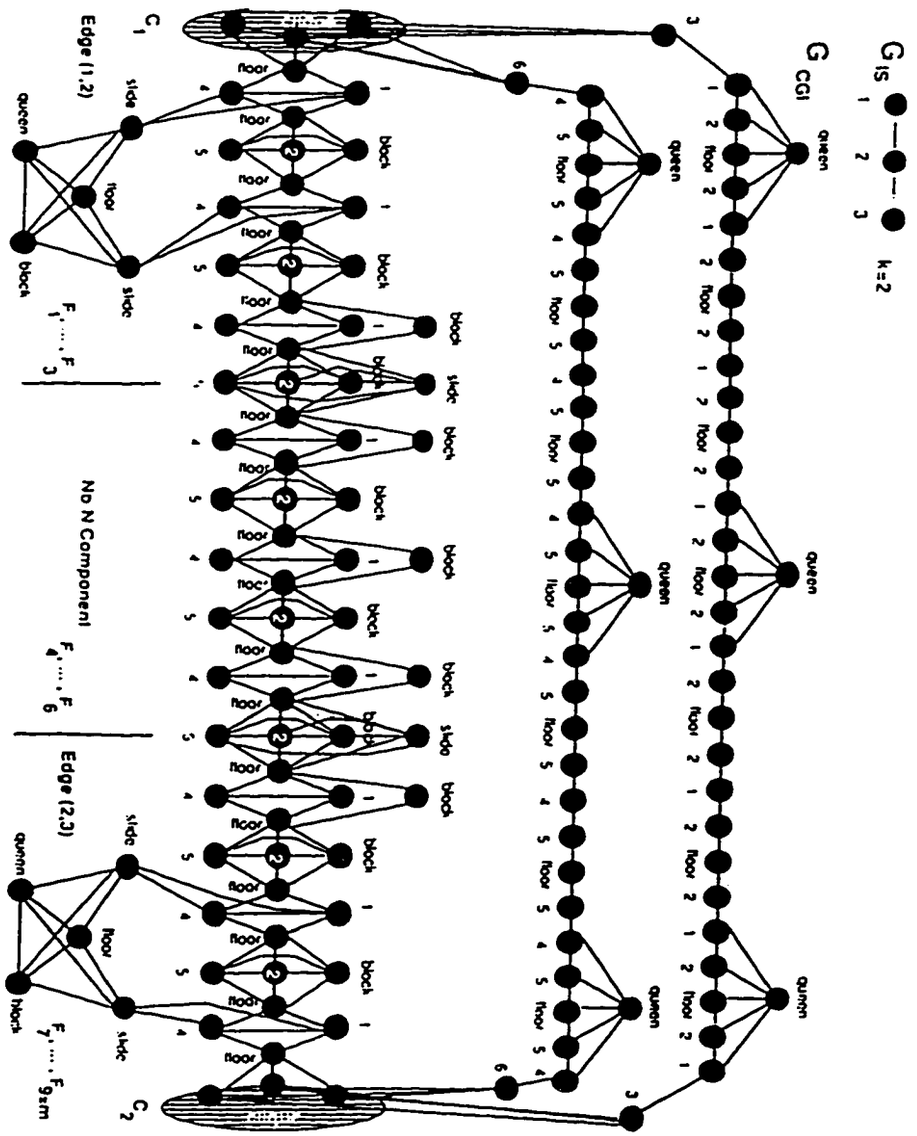

Figure 3.3: Example Construction of G_{ICG} from G_{IS}.

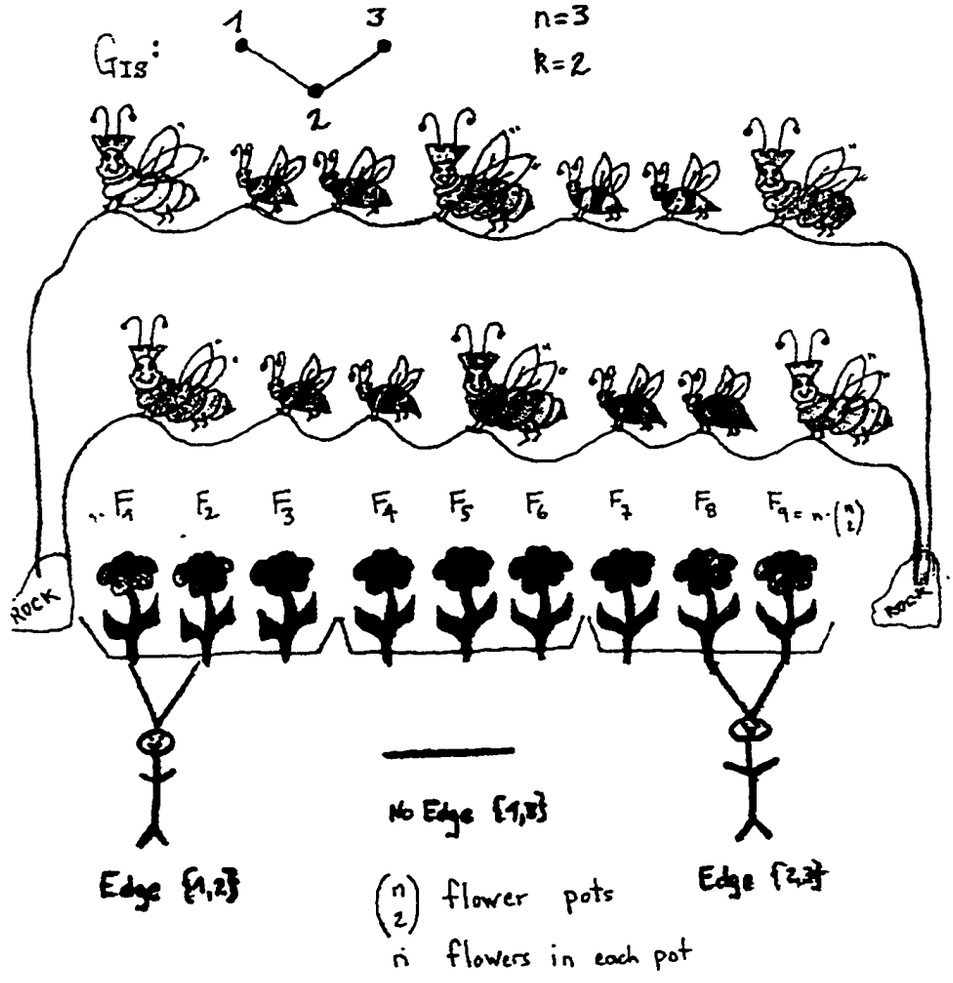

Figure 3.4: High-level description of the ICG reduction. The large bees represent elements of the set QB whilst small bees represent elements of set $B-QB$. There are n flowers (elements of set F) in each of the $\binom{n}{2}$ "flower pots". Each flower can have at most one queen bee land on it but any number of small bees. Each of the $|E|$ research assistants (*enforcer gadgets*) are assigned to watch for queen bees landing in two specific flowers (the end points of that edge) in their unique flower pot. If two queen bees arrive landing in both flowers (two vertices are chosen that have an edge between them), the research assistant becomes confused (this choice does not constitute an independent set for G_{IS}).

Golumbic, Kaplan and Shamir [57] proved $\mathcal{NP}$-completeness. Noticing that the redundancy factor in clone creation (corresponding to the size of the coloring k) was typically a small constant (typically between 5 and 10). Fellows, Hallett and Wareham began a parameterized analysis of ICG. Reported in the preceding two sections are proofs that ICG parameterized by the number of colors k is finite-state for the very restricted case of $k = 2$ but not FPT as k approaches infinity (hardness for $W[1]$).

Goldberg, Golumbic, Kaplan and Shamir [55, 57] consider a restricted variant of ICG (called UNIT ICG) where the resulting graph is required to be a *unit interval graph*: that is, an interval graph where every interval has the same length. Their model is based on the observation that while some cloning techniques generate clones of variable length, others produce clones of roughly the same length. It was hoped that this restricted version would allow for fast ($\mathcal{P}$-time or FPT) algorithms. Unfortunately, they found this problem to be $\mathcal{NP}$-complete and the parameterized (by k) version to be $W[1]$-hard. They did however present an $O(|V|^{k-1})$ algorithm to find a layout.

Later work by Bodlaender, Fellows and Hallett [17] raised the parameterized (by k) variant of ICG from being $W[1]$-hard to $W[t]$-hard, for all t. The combinatorics of the reduction in Section 3.4 were generalized and a large *system* of proofs was developed establishing $W[t]$-hardness, for all t, for a number of problems sharing the common trait of bounded width. See also [18].

Bodlaender and de Fluiter [21, 22] provide an alternative algorithm for ICG when $k = 2$ with a running time that is linear in the size of the graph G and a considerably more complicated (heroic) algorithm for $k = 3$ with running time $O(|V|^2)$. They also show that ICG is, in fact, $\mathcal{NP}$-complete for fixed $k \geq 4$. Their reduction is from THREE PARTITION, a problem not known to be hard for any level of the W-hierarchy, and hence it does not constitute a parameterized reduction.

The problem INTERVAL GRAPH COMPLETION (IG) which takes as input a graph G and asks whether there exists a set of at most p edges such that this supergraph is an interval graph is $\mathcal{NP}$-complete [54, 82]. Kaplan, Shamir and Tarjan [66] showed that this problem restricted to unit interval graphs is FPT when parameterized by the number of edges, p. (Table 3.1 refers to this problem as UNIT IG.) In terms of SEQUENCE RECONSTRUCTION, this result shows there exists a tractable algorithm when the number of additional overlap predictions necessary to reconstruct the sequence is a small number independent of the number of clones.

The first two authors, Kaplan and Shamir [67], proceeded to show that ICG parameterized by the number of colors k and the degree of the graph d is solvable in time $O(|V|^{d(k-1)})$ time. They also show a $O(|V|^{k(d-1)})$ time algorithm for the parameterization of ICG by both number of colors k and the degree d of the resulting graph G'. Although both algorithms have polynomial running times when k and d are fixed, the algorithms are not practical since the degree of the bounding would be very large. Hardness for any level of the W-hierarchy remains open for these problems. This parameterization is motivated by the observation that the graphs arising in practice tend to be sparse; that is, the number of clones any particular clone overlaps is between 8 and 10. (Bounding the pathwidth of a graph still allows for arbitrarily high degree vertices. Their model restricts all vertices to a constant degree.) Hardness for some level of the W-hierarchy remains open for both of these formulations. See Table 3.1 for a brief history of all of the above results.

ICG	Not finite state for $k \geq 3$	[49]
ICG	$O(\|G\|)$ algorithm for $k = 2$	[22, 49]
ICG	$\mathcal{NP}$-complete	[49, 57]
ICG	$W[1]$-hard (parameter k)[a]	[49]
UNIT ICG	$\mathcal{NP}$-complete	[55, 57]
UNIT ICG	$O(\|V\|^{k-1})$ algorithm	[65, 66]
UNIT ICG	$W[1]$-hard (parameter k)[a]	[65]
UNIT IG	FPT (parameter p)[b]	[66]
ICG	$W[t]$-hard, $\forall t$, parameter k	[17, 18]
ICG	$O(\|V\|^2)$ for $k = 3$	[21]
ICG	$\mathcal{NP}$-complete for fixed $k \geq 4$	[22]
ICG	$O(\|V\|^{d(k-1)})$ (parameter: k, d)[c]	[67]
ICG	$O(\|V\|^{k(d-1)})$ (parameter: k, d)[d]	[67]

Table 3.1: A history of INTERVALIZING COLORED GRAPHS and related problems.

[a] Where k is the number of colors.

[b] Where p is the maximum number of edge additions allowed.

[c] Degree of graph G is bounded by d.

[d] Degree of resulting graph G' is bounded by d.

Chapter 4

Shortest Common Supersequence

Sequence comparison is used by biologists and biochemists for several reasons. Similarity between a set of molecular sequences may indicate significant shared attributes between organisms the sequences represent. Or, from the similarity and disparity of these sequences, it may be possible to infer phylogenetic relationships amongst the organisms. If the sequences represent homologous genes, the comparison may be used to obtain information on molecular structure or function (see the survey paper [36] and references therein).

At the most abstract level, we may view a nucleic or amino acid sequence as a string formed over an alphabet of size 4 or 20, respectively. Computationally, multiple sequence comparison problems are viewed as string matching problems and have received much attention over the past twenty years (see [36, 52, 80] for excellent surveys of these results).

Historically, the following computational problem

> LONGEST COMMON SUBSEQUENCE
> **Instance:** A set of k strings $\{x_1, x_2, \ldots, x_k\}$ over an alphabet Σ, a positive integer m.
> **Question:** Is there a string $X \in \Sigma^*$ of length at least m that is a subsequence of x_i for $i = 1, \ldots, k$?

has played an important role in the design of algorithms modelling many multiple sequence comparison applications.

> MULTIPLE SEQUENCE ALIGNMENT (MSA)
> **Instance:** A set of molecular (nucleic or amino acid) sequences $X = \{x_1, x_2, \ldots, x_k\}$.

Output: An alignment $X' = \{x'_1, x'_2, \ldots, x'_k\}$ of the set X where (1) for some n, $|x'_i| = n$ for $1 \leq i \leq k$, (2) x'_i contains the sequence x_i and $n-$(length of x_i) copies of a special dummy base called an *indel* (denoted by Δ).

An alignment can be visualized as a $k \times n$ matrix in which row i corresponds to sequence i and column j corresponds to base j or to a special dummy "base" called an indel (denoted by Δ). A *gap* is a maximal string of indels within a row of an alignment. The *cost* of an alignment is usually computed relative to some function f which penalizes the alignment for various types of gaps and mismatches. Typically, biologists look for an alignment which minimizes the cost function f.

In fact, LCS is a restriction of the MULTIPLE SEQUENCE ALIGNMENT problem for the case where f is an arbitrary function providing a lower bound on the complexity for this version [16].

The second problem, CONSENSUS SUBSEQUENCE DISCOVERY, arises from the need to find short distinct characterizations of a set of molecular sequences. It is easy to see the equivalence of this problem with LCS.

CONSENSUS SUBSEQUENCE DISCOVERY (CSD)
Instance: A set of molecular (nucleic or amino acid) sequences $X = \{x_1, x_2, \ldots, x_k\}$.
Output: The longest sequence X' such that the base sequence of X' is a subsequence of each $x_i \in X$.

The LONGEST COMMON SUBSEQUENCE problem has been well-studied. There are many algorithms for the case where $k = 2$ (see the survey in [79]) and it is known to be $\mathcal{NP}$-complete over a binary alphabet and an arbitrary number of input strings [70]. In [73], they show LCS is $\mathcal{MAXSNP}$-hard for an arbitrary alphabet and [23] strengthens this result to alphabet size 2. This means that it does not have a fully polynomial time approximation scheme unless $\mathcal{P}=\mathcal{NP}$. The best asymptotic algorithms all have $\Theta(n^k)$ running times [58, 60].

In [16], LONGEST COMMON SUBSEQUENCE is parameterized in a number of meaningful ways. For all parameterizations with an unbounded alphabet, LCS is hard for various levels of the W-hierarchy. For fixed alphabet and parameterized by the length of the common subsequence, m, LCS is FPT. This is certainly "good news" for the CONSENSUS SUBSEQUENCE DISCOVERY problem if a small subsequence independent of the length of the input sequences suffices as a description for the k sequences. Otherwise, in the case that

larger subsequences are required, the variant of LCS with fixed alphabet and parameterized by the number of sequences k remains open.

Since the overwhelming majority of results concerning LONGEST COMMON SUBSEQUENCE indicate that tractable algorithms are unlikely, it seems reasonable to begin searching for alternative *measures* on multiple sequence similarity and disparity. The remaining portion of this chapter looks at a problem called SHORTEST COMMON SUPERSEQUENCE (SCS), a closely related problem to LONGEST COMMON SUBSEQUENCE. Intuitively, this problem seeks to find the shortest sequence having each input string as a subsequence. Since, in some sense, SCS and LCS provide *complementary* perspectives on the set of input strings, it should provide meaningful information to biologists. In the remainder of the chapter, we review known results and investigate the parameterized complexity of this problem.

4.1 History of SCS

As discussed in the previous section, LCS arose from molecular biology in the 1960s. Subsequently, related problems appeared having applications to text compression, data processing and mechanical engineering. SHORTEST COMMON SUPERSEQUENCE was one of these related problems (see [52, 62, 70, 79]). We define several parameterized variants below.

SHORTEST COMMON SUPERSEQUENCE (SCS).
Instance: Alphabet Σ, set of strings $R = \{r_1, r_2, \ldots, r_k\} \in \Sigma^*$, integer $M \in \mathbf{N}$.
Parameter: k. (SCS-1)
Parameter: M. (SCS-2)
Parameter: k, $|\Sigma|$. (SCS-3)
Question: Does there exist a string $S \in \Sigma^*$ of length at most M, that is a supersequence of each string in R?

Like LCS, the overwhelming majority of results pertaining to SCS are "negative" and the general picture for this problem is bleak. For fixed alphabet size 2, SCS is $\mathcal{NP}$-complete [70, 75, 79]. The best asymptotic algorithms are based upon dynamic programming and have running times of $O(kn^k)$ (this can be found in [79] and [4]). In [83], SCS is shown $\mathcal{MAXSNP}$-hard meaning it does not have a fully polynomial approximation scheme unless $\mathcal{P}=\mathcal{NP}$. This was later improved upon by [23] who show that it remains $\mathcal{MAXSNP}$-hard for a fixed alphabet size of 2. The best approximation algorithm for SCS guarantees a solution within a factor of 4 of optimal [63]. Table 4.1 lists the above results.

$\mathcal{NP}$-complete	$\|\Sigma\| = 5$	[70, 75]
$\mathcal{NP}$-complete	$\|\Sigma\| = 2$	[79]
$\mathcal{NP}$-complete	largest string length 2 orbit [a] is 3	[79]
$O(n^l)$	n =length of longest string l =number of given strings	[79]
$\mathcal{MAXSNP}$-hard	arbitrary alphabet	[83]
$\mathcal{MAXSNP}$-hard	fixed alphabet	[23]

Table 4.1: A history of SHORTEST COMMON SUPERSEQUENCE.

[a]Orbit is defined to be the maximum number of times a symbol appears in all of the input strings.

Given the difficulty of these problems, biologists and biochemists are constrained to measuring the commonality of a small number of sequences at a time. In practice, *the number of sequences (strings) is independent of the length of the strings and all other parameters of the problem.* This observation motivated the parameterized complexity analysis by Fellows, Hallett and Kirby in [48]. Here they proved two results specific to SHORTEST COMMON SUPERSEQUENCE and a number of other results pertaining to a set of related MINIMAL RUN SUPERSEQUENCE problems.

The first parameterization captures this notion of a bounded number of strings (SCS-1) and the theorem below strongly suggests that it is not FPT. In other words, parameterization by the number of strings does not reduce the overall asymptotic behaviour.

Theorem 13 ([48]) SHORTEST COMMON SUPERSEQUENCE PARAMETERIZED BY THE NUMBER OF INPUT STRINGS, k (SCS-1) is hard for $W[1]$. ∎

It is also noted in Fellows, Hallett, and Kirby [48] that SCS parameterized by the length of the overall supersequences (SCS-2) is tractable. However, this parameterization does not seem biologically relevant.

Theorem 14 ([48]) SHORTEST COMMON SUPERSEQUENCE PARAMETERIZED BY THE LENGTH OF THE OVERALL SUPERSEQUENCE, M (SCS-2) is fixed parameter tractable (FPT).

Proof: The algorithm is straightforward but we include it here for the sake of completeness. Define $\Sigma' = \{\sigma | \sigma \in \Sigma$ and σ appears in r for some $r \in R\}$. It must be the case that $|\Sigma'| \leq M$. Otherwise, answer NO. Now exhaustively check if one of the $O(M^{M+1})$ possible strings for M is a supersequence of all $r \in R$ simultaneously. ∎

Note that Theorem 13 allows for an arbitrary size alphabet when, in actual fact, the instances which are derived from biological applications will be formed from alphabets of a fixed size (either approximately 4 or 20 depending on the type of sequence). This observation provides another possible avenue for escaping the spectre of intractability. Section 4.3 is devoted to proving

Theorem 15: SCS PARAMETERIZED BY THE NUMBER OF STRINGS AND THE SIZE OF THE ALPHABET (SCS-3) is hard for the $W[1]$.

Of course, this does not rule out the possibility for tractable algorithms for a fixed size alphabet: it implies that as the number of strings and the size of the alphabet approaches infinity, SCS seems intractable.

4.2 Notation

An alphabet Σ is a finite set of symbols. A string over Σ is a finite sequence of symbols. Let $|S|$ denote the length of $S \in \Sigma^*$. Let $R \cdot S$ (or RS when no ambiguity can arise) denote the concatenation of string S to the end of string R. A subsequence R of a string S (write $R \in_s S$) is any string that can be obtained from S by deleting zero or more symbols from S. A string R is a supersequence of a string S if S is a subsequence of R. We call a string S a substring (or contiguous subsequence) of a string R if $R = S_1 S S_2$ for some strings S_1 and S_2. A string S is a supersequence of a set of strings $R = \{r_1, r_2, \ldots r_k\}$ if each r_i is a subsequence of S.

Let $R = r_1 r_2 \ldots r_k$ be a subsequence of $S = s_1 s_2 \ldots s_l$. An *embedding* of R in S is a monotonically increasing function f from $[1..k]$ to $[1..l]$ such that r_i and $s_{f(i)}$ are the same symbol for all $i \in [1..k]$. We say that r_i is mapped onto $s_{f(i)}$ by f. For a set $R = \{r_1, r_2, \ldots, r_k\}$ of strings and a string S an embedding of R in S is a k-tuple $(f_1, f_2, \ldots, f_k)$ where f_i is an embedding of r_i in S.

For a string S and $l \in \mathbb{N}$, define $(S)^l = (S) \cdot (S)^{l-1}$, $(S)^0 = \epsilon$, the empty string. Note that the i^{th} occurrence of symbol $x \in \Sigma$ in string S is the element $x \in_s S$ such that $S = S_1 x S_2$

and $(x)^{i-1} \in_s S_1$ but $(x)^i \notin_s S_1$, for $i \geq 1$.

4.3 Hardness for SCS-3

We reduce from the problem CLIQUE which is proven $W[1]$-complete in [41].

k–CLIQUE
Instance: A finite graph $G = (V, E)$ with no self-loops, an integer k.
Parameter: k.
Question: Does there exist a set $V' \subseteq V$, $|V'| \geq k$, such that, for all $u, v \in V'$, $(u, v) \in E$?

Theorem 15: SHORTEST COMMON SUPERSEQUENCE PARAMETERIZED BY THE NUMBER OF STRINGS k AND THE SIZE OF THE SIZE OF THE ALPHABET $|\Sigma|$ (SCS-3) is hard for $W[1]$.

Let $G = (V, E)$ and k be an instance of the k–CLIQUE problem. W.l.o.g. assume the vertices are labeled by integers from $\{1 : |V|\}$. For edges (x, y) and (x', y') in E, we say that $(x, y) \leq (x', y')$, if either $x < x'$ or $x = x'$ and $y \leq y'$. Let $\sigma : E \to \{1 : |E|\}$ be the bijection defined as follows: $(x, y) \leq (x', y')$ if and only if $\sigma((x, y)) \leq \sigma((x', y'))$. As a notational convenience we simply write $\sigma(x, y)$ to denote $\sigma((x, y))$.

We create an instance of the SHORTEST COMMON SUPERSEQUENCE problem as follows. For $i, j, 1 \leq i < j \leq k$, let Σ be the following set of (arbitrary) symbols : $\{L^i_{i,j}, L^j_{i,j}, P^i_{i,j}, P^j_{i,j}, I^i_{i,j}, I^j_{i,j}\} \cup \{\langle \mathsf{b} \rangle, \langle \mathsf{f} \rangle\}$.

$$
\begin{aligned}
Z_{up} &= (\langle \mathsf{b} \rangle \cdot \langle \mathsf{f} \rangle)^{|E|-1} \cdot \langle \mathsf{b} \rangle \cdot \mathrm{UpSelector} \cdot \langle \mathsf{f} \rangle \cdot (\langle \mathsf{b} \rangle \cdot \langle \mathsf{f} \rangle)^{|E|-1} \\
\mathrm{UpSelector} &= \prod_{i=1}^{k} \mathrm{Select}(i) \\
\mathrm{Select}(i) &= \mathrm{Latch}(i) \cdot (\mathrm{Position}(i))^{|V|+1} \cdot \mathrm{Latch}(i) \\
\mathrm{Latch}(i) &= (\prod_{j=1}^{i-1} L^i_{j,i}) \cdot (\prod_{j=i+1}^{k} L^i_{i,j}) \\
\mathrm{Position}(i) &= (\prod_{j=1}^{i-1} P^i_{j,i}) \cdot (\prod_{j=i+1}^{k} P^i_{i,j}) \\
\\
Z_{down} &= (\langle \mathsf{b} \rangle \cdot \langle \mathsf{f} \rangle)^{|E|-1} \cdot \langle \mathsf{b} \rangle \cdot \mathrm{DownSelector} \cdot \langle \mathsf{f} \rangle \cdot (\langle \mathsf{b} \rangle \cdot \langle \mathsf{f} \rangle)^{|E|-1}
\end{aligned}
$$

$$\text{DownSelector} = \prod_{i=1}^{k}(\text{Latch}(i) \cdot \text{InterTwine}(i) \cdot \text{Latch}(i))$$

$$\text{InterTwine}(i) = ((\prod_{j=1}^{i-1} I^i_{j,i}) \cdot (\prod_{j=i+1}^{k} I^i_{i,j}))^2$$

Let $X = \{X_{i,j} | 1 \leq i < j \leq k\}$ where

$$X_{i,j} = \prod_{p=1}^{|E|}(\langle \text{b} \rangle \cdot \text{EdgeEnforcer}(\sigma^{-1}(p), (i,j)) \cdot \langle \text{f} \rangle)$$

$$\text{EdgeEnforcer}((u,v),(i,j)) = L^i_{i,j} \cdot \text{EndPt}(i,u,(i,j)) \cdot L^i_{i,j} \cdot L^j_{i,j} \cdot \text{EndPt}(j,v,(i,j)) \cdot L^j_{i,j}$$

$$\text{EndPt}(x,y,(i,j)) = (\prod_{q=1}^{y} P^x_{i,j}) \cdot I^x_{i,j} \cdot I^x_{i,j} \cdot (\prod_{q=y}^{|V|} P^x_{i,j})$$

Note that $|R| = \binom{k}{2} + 2$ and $|\Sigma| = 6\binom{k}{2} + 2$. Let M, the length of the common supersequence, equal $4|E| + |E|\binom{k}{2}(2|V| + 10) - 2$.

Figures 4.1 and 4.2 give an example construction of an instance of SCS-3 and several of the combinatorial gadgets described above. We interpret "L" symbols from Σ as representing *latches*, P symbols as representing *postion*, and I symbols as representing *intertwine*. Strings Z_{up} and Z_{down} act as *selection* components and provide a *backbone structure* for the $X_{i,j}$ strings to *latch* on to. We typically use i, j to index the set X and u, v to represent elements of $V(G)$.

Lemma 2: G has a clique of size k if and only if R has a common supersequence of length $\leq M$.

Proof: ($\Rightarrow$) Let G have a clique of size k labelled $\{u_1, u_2, \ldots, u_k\}$. Since $(u_i, u_j) \in E$, there exists a contiguous subsequence $(\langle \text{b} \rangle \cdot \text{EdgeEnforcer}((u_i, u_j),(i,j)) \cdot \langle \text{f} \rangle)$ in each $X_{i,j}$. Let $m_{i,j} = \sigma(u_i, u_j)$ for all i, j, $1 \leq i < j \leq k$. Let $\mathcal{S}$ be the common supersequence of R and construct $\mathcal{S}$ as follows:

0. $\mathcal{S} \leftarrow \epsilon$

   for $p : 1..|E| - 1$ **do**

1. $\mathcal{S} \leftarrow \mathcal{S} \cdot \langle \text{b} \rangle \cdot (\prod_{i=1}^{k-1} \prod_{j=i+1}^{k} \begin{cases} \epsilon & \text{if } p \geq m_{i,j} \\ \text{EdgeEnforcer}(\sigma^{-1}(p),(i,j)) & \text{otherwise} \end{cases}) \cdot \langle \text{f} \rangle$

Figure 4.1: Example Construction of an instance of SCS-3 from G_{Clique}.

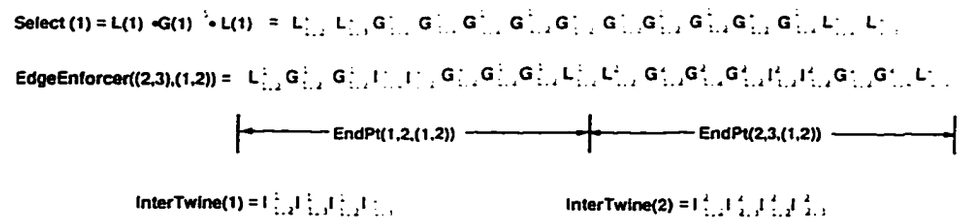

Figure 4.2: Combinatorial Gadgets used in the SCS-3 reduction

end for

2. $\mathcal{S} \leftarrow \mathcal{S} \cdot \langle \mathrm{b} \rangle$

for $p : 1..k$ **do**

3. $\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \mathrm{Latch}(p)$

for $r : 1..u_p$ **do**

4. $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \mathrm{Position}(p)$

end for

5. $\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \mathrm{InterTwine}(p)$

for $r : u_p..|V|$ **do**

6. $\quad\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \mathrm{Position}(p)$

end for

7. $\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \mathrm{Latch}(p)$

end for

8. $\mathcal{S} \leftarrow \mathcal{S} \cdot \langle \mathrm{f} \rangle$

for $p : 2..|E|$ **do**

9. $\quad \mathcal{S} \leftarrow \mathcal{S} \cdot \langle \mathrm{b} \rangle \cdot (\prod_{i=1}^{k-1} \prod_{j=i+1}^{k} \begin{cases} \epsilon & \text{if } p < m_{i,j} \\ \mathrm{EdgeEnforcer}(\sigma^{-1}(p), (i,j)) & \text{otherwise} \end{cases}) \cdot \langle \mathrm{f} \rangle$

end for

Claim 2.1: (i) $|\mathcal{S}| = M$ and (ii) $r \in_s \mathcal{S}$. for all $r \in R$.

Proof: (i) To see that $|S| = M$, note that lines 1, 2, 8, 9 contribute $4|E|-2$ $\langle \mathrm{b} \rangle$ and $\langle \mathrm{f} \rangle$ symbols. Lines 1, 9 contribute $(|E| - 1)\binom{k}{2}(2n + 10)$ symbols from the EdgeEnforcer gadgets. Lines 3, 7 contribute a total of $k(k-1)2$ "L" symbols whilst lines 4, 6 provide $k(k-1)(n+1)$ "P" symbols. Line 5 provides $k(k-1)2$ "I" symbols. It is straightforward to verify this summation equals M.

(ii) There are 3 cases.

Case 1: $Z_{up} \in_s \mathcal{S}$. It is easy to verify that lines 1, 2, 8, 9 supply the correct order of ⟨b⟩ and ⟨f⟩ symbols. Lines 3, 4, 6, 7 provide the correct order of symbols to construct the Select(1), ..., Select(k) gadgets.

Case 2: $Z_{down} \in_s \mathcal{S}$. Again, it is easy to verify that lines 1, 2, 8, 9 supply the correct order of ⟨b⟩ and ⟨f⟩ symbols. Lines 3, 7 supply the correct order of symbols to construct the Latch gadgets and line 5 supplies the InterTwine gadgets.

Case 3: $X_{i,j} \in_s \mathcal{S}$, for all i, j, $1 \leq i < j \leq k$. Line 1 places EdgeEnforcer($\sigma^{-1}(p), (i, j)$) between the p–th occurrence of ⟨b⟩ in $\mathcal{S}$ and the p–th occurrence of ⟨f⟩ in $\mathcal{S}$ for $p < m_{i,j}$. To see that EdgeEnforcer($\sigma^{-1}(m_{i,j}), (i, j)$) is a subsequence of $\mathcal{S}$, note that lines 4, 5, 6 place InterTwine(i) between the m_i–th occurrence of Position(i) and the $(m_i + 1)$–th occurrence of Position($i + 1$), for $(m_i, m_j) = \sigma^{-1}(m_{i,j})$. Also, InterTwine($j$) is placed between the m_j–th occurrence of Position(j) and the $(m_j + 1)$–th occurrence of Position(j). Lines 3, 7 supply the appropriate sequences for the Latch gadgets. Line 9 placed EdgeEnforcer($\sigma^{-1}(p), (i, j)$) between the $(|E| + p)$–th occurrence of ⟨b⟩ and the $(|E| + p)$–th occurrence of ⟨f⟩.

This establishes the claim and the forward direction of Lemma 2. ∎

(⇐) Let R have a common supersequence $\mathcal{S}$ of length $\leq M$. We begin by showing a simple lower bound on the length of $\mathcal{S}$. Note that there are $\geq 4|E| - 2$ ⟨b⟩ and ⟨f⟩ symbols in $\mathcal{S}$ since $(\langle b \rangle \cdot \langle f \rangle)^{2|E|-1} \in_s Z_{up}$. Define $X'_{i,j}$ to be the subsequence of $X_{i,j}$ by deleting all occurrences of the ⟨b⟩ and ⟨f⟩ symbols. For two distinct strings $X'_{i,j}$ and $X'_{i',j'}$, there is no symbol from Σ that appears in both of these strings simultaneously. This implies any common supersequence of $\cup_{1 \leq i < j \leq k} X'_{i,j}$ (and therefore X) has length $\geq \binom{k}{2}|E|(2|V| + 10)$. In total, any common supersequence for X and the ⟨b⟩ and ⟨f⟩ symbols from Z_{up} has length $\geq 4|E| - 2 + \binom{k}{2}|E|(2|V| + 10) = M$.

The next two statements follow directly from the preceding argument.

Corollary 3: Every ⟨b⟩ or ⟨f⟩ symbols in any $X_{i,j}$ or Z_{down} must be mapped to the same element of $\mathcal{S}$ as a ⟨b⟩ or ⟨f⟩ symbol from Z_{up}. ∎

Corollary 4: For every symbol in the UpSelector or DownSelector gadgets of Z_{up} and Z_{down}, there is a symbol in some $X_{i,j}$ such that both are mapped to the same element of $\mathcal{S}$. ∎

It is easy to see that the p–th $\langle$b$\rangle$ symbol (p–th $\langle$f$\rangle$ symbol) from Z_{up} and the p–th $\langle$b$\rangle$ symbol (p–th $\langle$f$\rangle$ symbol) from Z_{down} must be mapped to the same element of $\mathcal{S}$.

Fix i and j, $1 \leq i < j \leq k$. Let Select$'(i)$ be the subsequence of Select(i) obtained by deleting all symbols of the form $L^i_{i',j'}$ and $P^i_{i',j'}$, where $i' \neq i$ or $j' \neq j$. Now consider $X_{i,j}$. By Corollary 4, every symbol of Select$'(i)$ must be mapped to the same element of $\mathcal{S}$ as some subsequence χ of $X_{i,j}$.

Claim 4.1: χ is a contiguous subsequence of $X_{i,j}$.

Proof: Suppose otherwise. Write $\chi = \chi_1, \chi_2, \ldots, \chi_r$ so that each χ_q is a contiguous subsequence in $X_{i,j}$. It follows that there exists a q such that χ_q is a contiguous subsequence in EdgeEnforcer$(\sigma^{-1}(s), (i,j))$ and χ_{q+1} is a contiguous subsequence in EdgeEnforcer$(\sigma^{-1}(t), (i,j))$ for $1 \leq s < t \leq |E|$. In the common supersequence $\mathcal{S}$, there exists $(t-s)$ $(\langle$f$\rangle \cdot \langle$b$\rangle)$ sequences of symbols from $X_{i,j}$ not mapped to the same elements of $\mathcal{S}$ as $(t-s)$ subsequences of $(\langle$f$\rangle \cdot \langle$b$\rangle)$ symbols from Z_{up}. This contradicts Corollary 3. ∎

Now χ is a contiguous subsequence of $X_{i,j}$ and note that χ must be equal to one of the $|E|$ $L^i_{i,j} \cdot \text{EndPt}(i, u, (i,j)) \cdot L^i_{i,j}$ gadgets from $X_{i,j}$. The next claim follows directly from this observation and how the $X_{i,j}$ were constructed.

Claim 4.2: The subsequence $L^i_{i,j} \cdot L^i_{i,j}$ of EndPt$(i, u, (i,j))$ is between the u–th occurrence of $P^i_{i,j}$ and the $(u+1)$–th occurrence of $P^i_{i,j}$ from Z_{up}, for some $1 \leq u \leq |V|$.

We call the subsequence defined by these gadgets to be the *signature* for vertex u.

Define Select$'(j)$ in the same manner as Select$'(i)$ but deleting symbols of the form $L^j_{i',j'}$ and $P^j_{i',j'}$. By the same argument as in Claim 4.1, we can show there exists a χ' corresponding to a contiguous subsequence of the form $L^j_{i,j} \cdot \text{EndPt}(j, v, (i,j)) \cdot L^j_{i,j}$.

Claim 4.3: $\chi \cdot \chi'$ is a contiguous subsequence of $X_{i,j}$.

Proof: Suppose otherwise. Then χ is a contiguous subsequence of EdgeEnforcer$(\sigma^{-1}(p), (i,j))$ whilst χ' is a contiguous subsequence of EdgeEnforcer$(\sigma^{-1}(q), (i,j))$ for $p < q$. Therefore, there exists $(q-p)$ $\langle$f$\rangle \cdot \langle$b$\rangle$ sequences of symbols not mapped to the same elements of $\mathcal{S}$ as $(q-p)$ $\langle$f$\rangle$ and $(q-p)$ $\langle$b$\rangle$ symbols from Z_{up}. Again, this contradicts Corollary 3. ∎

Observe that the contiguous subsequence $\chi \cdot \chi'$ must be exactly EdgeEnforcer$((u,v), (i,j))$ for some $(u,v) \in E$. We say that $X_{i,j}$ *enforces* $(u,v) \in E$.

Claim 4.4: If $X_{i,j}$ *enforces* edge $(u, v) \in E$ then,
(i) $X_{i,j'}$, $j' \neq j$, *enforces* some edge of the form $(u, v') \in E$ for $u < v' \leq |V|$, and
(ii) $X_{i',j}$, $i' \neq i$, *enforces* some edge of the form $(u', v) \in E$ for $1 \leq u' < v$.

Proof: Let $X_{i,j}$ *enforce* edge $(u, v) \in E$.
(i) Suppose this were not true and $X_{i,j'}$ *enforces* some edge $(u', v') \in E$, $u' \neq u$. W.l.o.g. assume $u' > u$. Then, by Claim 4.2, the subsequence $I^i_{i,j} \cdot I^i_{i,j}$ of $X_{i,j}$ is between the u–th occurrence of $P^i_{i,j}$ and the $(u+1)$–th occurrence of $P^i_{i,j}$. Again, by Claim 4.2, the subsequence $I^i_{i,j'} \cdot I^i_{i,j'}$ of $X_{i,j'}$ is between the u'–th occurrence of $P^i_{i,j'}$ and the $(u'+1)$–th occurrence of $P^i_{i,j'}$.

Now $I^i_{i,j} \cdot I^i_{i,j'} \cdot I^i_{i,j} \cdot I^i_{i,j'}$ is a subsequence of InterTwine(i). But $I^i_{i,j} \cdot I^i_{i,j}$ appears before the $(u+1)$–th occurrence of $P^i_{i,j}$ implying they appear before the u'–th occurrence of $P^i_{i,j'}$. Since $I^i_{i,j'} \cdot I^i_{i,j'}$ appears after the u'–th occurrence of $P^i_{i,j'}$, at least one symbol of the InterTwine(i) gadget is not mapped to the same element of $\mathcal{S}$ as any element appearing in any of the strings of X. This contradicts Corollary 4.

(ii) An argument symmetric to that of part (i) establishes this case as well. ∎

Claim 4.5: The set X *enforces* $\binom{k}{2}$ distinct edges.

Proof: Suppose $X_{i,j}$ and $X_{i',j'}$ both *enforce* edge $(u_p, u_q) \in E$, $i \neq i'$ or $j \neq j'$. W.l.o g., assume $j \neq j'$. Consider $X_{j,j'}$. By Claim 4.4, it must *enforce* an edge of the form (u_q, u_q). But G is a graph without self-loops. $(u_q, u_q) \notin E$, and hence there is no $\langle \text{b} \rangle \cdot \text{EdgeEnforcer}((u_q, u_q), (j, j')) \cdot \langle \text{f} \rangle$ in $X_{j,j'}$. Contradiction. ∎

Claim 4.6: The contiguous subsequences Select$(i) \in_s Z_{up}$ and InterTwine$(i) \in Z_{down}$, $1 \leq i \leq k$, define k distinct *signatures* $\{u_1, u_2, \ldots, u_k\}$.

Proof: Suppose there are $< k$ distinct signatures defined by $\mathcal{S}$. Then there exists i and i', $i \neq i'$, such that both Select(i), InterTwine(i) and Select(i'), InterTwine(i') define the signatures for vertex $u_i = u_{i'}$. Now consider string $X_{i,i'}$. It must *enforce* an edge of the form (u_i, u_i), but G is a graph without self-loops. $(u_i, u_i) \notin E$, hence there is not gadget of the form $\langle \text{b} \rangle \cdot \text{EdgeEnforcer}((u_i, u_i), (i, i')) \cdot \langle \text{f} \rangle$ in $X_{i,i'}$. Contradiction. ∎

The $\{u_1, u_2, \ldots, u_k\}$ correspond to k distinct mutually adjacent vertices in G and now the backwards direction of Lemma 2 follows. ∎

Chapter 5

Open Problems and Conclusions

We feel that the major contributions of the work presented herein are:

- A thorough complexity analysis from the vista of various computational complexity frameworks of the problem INTERVALIZING COLORED GRAPHS. Our combinatorial problem models in a straightforward way the determination of contig assemblies in the mapping of DNA via restriction enzyme based approaches. This conceptual simplicity allows for ease of study from theoretical perspectives. A mapped of the inherent hardness of this problem is presented and several shortcomings and weaknesses are discussed.
- We rule out the possibility of fixed parameter tractable algorithms for SHORTEST COMMON SUPERSEQUENCE unless unlikely collapses occur. This can be seen as a failed attempt to circumvent the intractability one faces when attempting to align multiple molecular sequences, a problem arising often in biology and biochemistry.
- Via the $\star$–heuristic, we find an interesting relationship between the combinatoric mechanics of non finite-stateness proofs and parameterized reductions. This gives a useful tool for the initial phases of parameterized reduction design.
- In the case of k–INTERVALIZING COLORED GRAPHS, we provide a proof technique for showing hardness for the W–hierarchy which is technically demanding but conceptually simple.

For INTERVALIZING COLORED GRAPHS and SHORTEST COMMON SUPERSEQUENCE it remains open whether these problems belong to some level of the hierarchy. Currently, it

is known they are hard for $W[t]$, for all t, and hard for $W[t]$ respectively. In the former case, membership in $W[P]$ implies $\mathcal{P} = \mathcal{NP}$ and; therefore, it seems unlikely this problem belongs to even this top-most region. Furthermore, it would seem reasonable that ICG is hard for $W[P]$ but no proof has been forthcoming.

A proof of membership for SHORTEST COMMON SUPERSEQUENCE does not carry with it such drastic complexity collapses: although, once again, no proof of membership nor hardness for $W[P]$ has been found.

Besides INTERVALIZING COLORED GRAPHS and SHORTEST COMMON SUPERSEQUENCES, many other problems such as BANDWIDTH, PRECEDENCE CONSTRAINED k-PROCESSOR SCHEDULING, and TRIANGULATING COLORED GRAPHS have been show hard for some level of the W-hierarchy yet their membership at any level remains open. This hints at interesting classes above $W[P]$ which have been for the most part still unexplored.

Of course, all the variations of ICG presented thus far have unacceptably bad asymptotic running times. Possibly a more refined model of the SEQUENCE RECONSTRUCTION problem will lead to more tractable problems. There is some hope for this and a more thorough examination of actual data arising from experimental biology may be the key to such a breakthrough. For example, it may be feasible to characterize better the family of graphs induced by the biological data or possibly to understand better where and in what form errors manifest themselves in the data.

Our results show that the computational complexity of INTERVALIZING COLORED GRAPHS varies dramatically with the redundancy factor k. It seems natural to consider models of the SEQUENCE RECONSTRUCTION problem which do not require that the assembled sequence to be one single contig but, instead, a small set of contigs which could be joined together later using separate biological experiments. Lowering the redundancy factor to say 3 would allow the use of known polynomial time algorithms for ICG and still provide relatively few disconnected contigs (see [74] for a preliminary discussion of this issue).

Two interesting open problems which seem very biologically relevant are determining the complexity of LONGEST COMMON SUBSEQUENCE and SHORTEST COMMON SUPERSEQUENCE when the size of the alphabet Σ is fixed and the number of input strings is a parameter. The cases where the size of Σ is approximately 4 to 6 (for nucleic acid sequences) and 16 to 20 (for amino acid sequences) are of particular interest.

Bibliography

[1] K. A. Abrahamson, R. G. Downey and M. R. Fellows. Fixed-parameter intractability II. *Proc. 10th Symposium on Theoretical Aspects of Computer Science (STACS '93)*. 1993, pp. 374–385.

[2] K. A. Abrahamson, R. G. Downey and M. R. Fellows. Fixed parameter tractability and completeness IV : on completeness for $W[P]$ and $PSPACE$ analogues. *Annals of Pure and Applied Logic*, 75(1995). 235-276.

[3] K. A. Abrahamson and M. R. Fellows. Finite Automata, Bounded Treewidth and Well-Quasiordering. *Contemporary Mathematics*. 147(1993), 539–563.

[4] A. V. Aho, D. S. Hirschberg and J. D. Ullman. Bounds on the complexity of the longest common subsequence problem. *J. ACM*. 23.1(1976), 1–12.

[5] F. Alizadeh, R. M. Karp, L. E. Newberg and D. K. Weisser. Physical mapping of chromosomes: a combinatorial problem in molecular biology. *Algorithmic*, 13(1995).52–76.

[6] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability: a survey. *BIT*. 25(1985),2-23.

[7] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(1991),308–340.

[8] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23(1984),11–24.

[9] R. Balasubramanian, M. R. Fellows and V. Raman. An improved fixed parameter algorithm for vertex cover. To appear. Dept. of Computer Science, University of Victoria. manuscript, 1995.

[10] M. J. Bishop and C. J. Rawlings (eds). Nucleic acid and protein sequence analysis: a practical approach. *IRL Press Ltd.*, Oxford. 1987.

[11] H. L. Bodlaender. Dynamic programming algorithms on graphs with bounded treewidth. *Proc. 15th International Colloquium on Automata, Languages and Programming (ICALP '88)*. Springer Verlag, Lecture Notes in Computer Science. 317(1988),103–118.

[12] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *Proc. 25th Annual ACM Symposium on Theory of Computing (STOC '93)*.1993. pp. 226–234.

[13] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*. 11(1993), 1–23.

[14] H. L. Bodlaender. A partial k–arboretum of graphs with bounded treewidth. To appear. 1996.

[15] H. L. Bodlaender. R. G. Downey, M. R. Fellows, M. T. Hallett and H. T. Wareham. Parameterized complexity analisys in computational biology. *CABIOS*. 11.1(1994),49–57.

[16] H. L. Bodlaender. R. G. Downey, M. R. Fellows and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theor. Comp. Sc.*. 147(1994),31–54.

[17] H. L. Bodlaender. M. R. Fellows and M. T. Hallett. The parameterized complexity of DNA Mapping. Perfect Phylogeny and other problems of 'bounded width' (extended abstract). *Proc. 26th Annual ACM Symposium on the Theory of Computing (STOC '94)*.1994. pp. 449–458.

[18] H. L. Bodaender. M. R. Fellows, M. T. Hallett, H. T. Wareham and T. Warnow. The hardness of thin colored graphs. Submitted to *SIAM J. Comp*. 1995.

[19] H. L. Bodlaender. M. R. Fellows and M. T. Hallett. The parameterized intractability of bandwidth and related problems. To be submitted. Manuscript. 1996.

[20] H. L. Bodlaender. M. R. Fellows and T. Warnow. Two Strikes Against Perfect Phylogeny. *Proc. 19th International Colloquium on Automata, Languages and Programming (ICALP '92)*, Springer Verlag, Lecture Notes in Computer Science, 623(1992).273–283.

[21] H. L. Bodlaender and B. de Fluiter. Intervalizing k-colored graphs. *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP '95)*. Springer-Verlag. Lecture Notes in Computer Science, 944(1994).87–98.

[22] H. L. Bodlaender and B. de Fluiter. Intervalizing k–colored graphs for DNA physical mapping. Tech. Report, UU-CS-1995-20, Universiteit Utrecht. Department of Computer Science. Utrecht, The Netherlands, 1995.

[23] P. Bonnizoni. M. Duella and G. Mauri. SCS and LCS are $MAX-SNP$ hard over a binary alphabet. Technical Report 117/94, Department of Computer Science, University of Milan. 1994.

[24] R. B. Borie. R. G. Parker and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*. 7(1992),555–582.

[25] S. Buss. In a private communication with M. R. Fellows and R. G. Downey.

[26] L. Cai and J. Chen. On the amount of nondeterminsim and the power of verifying. *Proc. International Conference on Mathematical Foundations of Computer Science (MFCS '93)*, pp. 311–320. 1993.

[27] L. Cai and J. Chen. On fixed-parameter tractability and approximation of NP hard optimization problems. *Proc. Israel Conf. on Theoretical Computer Science (ISTCS '93)*. pp. 118–126. 1993.

[28] L. Cai. J. Chen. R.G. Downey and M.R. Fellows. The parameterized complexity of short computation and factorization. Technical Report, Department of Computer Science. University of Victoria. 1993.

[29] L. Cai, J. Chen. R. G. Downey and M. R. Fellows. On the structure of parameterized problems in NP. *Proc. 11th Annual Symposium on the Theoretical Aspects of Computer Science (STACS '94)*. Springer-Verlag, Lecture Notes in Computer Science.1994. pp. 509–520.

[30] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. Technical report. Department of Computer Science, The Chinese University of Hong Kong. Shatin. New Territories, Hong Kong, 1995. To appear: *Information Processing Letters.*

[31] P. Cholak and R. G. Downey. Undecidability and definability for parameterized polynomial time m-reducibilities. In: J. Crossley, J. Remmel, R. Shore and M. Sweedler (eds), *Logical Methods*. Birkhauser, Boston (1994), pp. 194–221.

[32] S. A. Cook. The complexity of theorem-proving procedures. *Proc. 3rd. Annual Symp. on Theory of Computing*. (1971), pp. 151–158.

[33] N. G. Cooper (editor). The human genome project. *Los Alamos Science*. 20.1992.

[34] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85(1990),12–75.

[35] W. H. E. Day and F. R. McMorris. The computation of consensus patterns in DNA sequences. *Mathematical and Computer Modelling*, 17.10(1993),49–52.

[36] W. H. E. Day and F. R. McMorris. Alignment, comparison and consensus of molecular sequences. In: E. Diday, Y. Lechevallier, M. Schader, P. Bertrand and B. Burtschy (eds), *New Approaches in Classification and Data Analysis*. Springer-Verlag, (1993). pp. 347–355.

[37] R. G. Downey, P. Evans and M. R. Fellows. Parameterized learning theory. *Proc. 6th Annual Conference on Learning Theory (COLT '93)*. ACM Press, New York, 1993. pp. 51–57.

[38] R. G. Downey and M. R. Fellows. Fixed-parameter intractability (extended abstract). *Proc. Seventh Annual Conference on Structure in Complexity Theory (Structures '92)*,1992, pp. 36–49.

[39] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness III: some structural aspects of the W-hierarchy. In: K. Ambos-Spies, S. Homer and U. Schoning (eds), *Complexity Theory*. Cambridge University Press.1993. pp. 166–191.

[40] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(1995),873-921.

[41] R. G. Downey and M. R. Fellows. Fixed parameter tractability and completeness II : On completeness for $W[1]$. *Theoretical Computer Science*, 141(1995),109–131.

[42] R. G. Downey and M. R. Fellows. Parameterized computational feasibility. *Proc. of Feasible Mathematics (Feasible Mathematics II)*. P. Clote, J. B. Remmel (eds), Birkhauser, Boston, 1995, pp. 219–244.

[43] R. G. Downey and M. R. Fellows. Parameterized complexity. Manuscript. To appear in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[44] R. G. Downey, M. R. Fellows, B. M. Kapron, M. T. Hallett and H. T. Wareham. The parameterized complexity of some problems in logic and linguistics. *Proc. Symposium on Logical Foundations of Computer Science (LFCS '94)*, Springer-Verlag, Lecture notes in Computer Science, 813(1994),89–100.

[45] R. G. Downey, M. R. Fellows and K. Regan. Parameterized circuit complexity and the W-hierarchy. To apear. Dept. of Computer Science, University of Victoria, 1995.

[46] M. Doyle. personnal communication. 1996.

[47] G. A. Evans. Combinatoric strategies for genome mapping. *Bioassays*, 13(1991),39–44.

[48] M. R. Fellows, M. T. Hallett and D. Kirby. The parameterized complexity of shortest common supersequence. Manuscript. 1995.