Parameterized Enumeration of Neighbour Strings and Kemeny Aggregations

by

Narges Simjour

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2013

Abstract

In this thesis, we consider approaches to enumeration problems in the parameterized complexity setting. We obtain competitive parameterized algorithms to enumerate all, as well as several of, the solutions for two related problems Neighbour String and Kemeny Rank Aggregation. In both problems, the goal is to find a solution that is as close as possible to a set of inputs (strings and total orders, respectively) according to some distance measure.

We also introduce a notion of *enumerative* kernels for which there is a bijection between solutions to the original instance and solutions to the kernel, and provide such a kernel for Kemeny Rank Aggregation, improving a previous kernel for the problem.

We demonstrate how several of the algorithms and notions discussed in this thesis are extensible to a group of parameterized problems, improving published results for some other problems.

Acknowledgements

I would like to thank my supervisor, Professor Naomi Nishimura, for her generous support and invaluable advice on my research. I would also like to thank Professor Jonathan Buss and Professor Timothy Chan for their helpful comments on the direction of my research. I wish to thank Professor Bin Ma for fruitful discussions on some of the results in this work. I am also grateful to my thesis committee for spending their valuable time reading this thesis, and for their suggestions which improved the content and presentation of my work.

Special thanks to my family for their encouragement and love, and to the many friends I met in Waterloo, for making my PhD experience really enjoyable.

Dedication

This is dedicated to my husband, Ehsan.

Table of Contents

List of Figures

Chapter 1

Introduction

In many problems, enumeration of multiple, or all, the solutions is preferred to computing a single solution. This is particularly useful in situations where the problem defined is a simplification of what is required, and finding multiple solutions increases the chance of finding an ideal solution. Enumeration algorithms for a problem can also be useful for computing a single solution to other problems. Examples include algorithms for EDGE DOMINATING SET [67, 132, 72, 23, 131], MINIMUM-WEIGHT MAXIMAL MATCHING [72, 131], and CONNECTED VERTEX COVER [83, 69, 23], all of which use an enumeration of all minimal vertex covers, and construct their solutions based on one of the minimal vertex covers. Another example is Marx's algorithm [108] for certain constraint satisfaction problems, which exploits an enumeration of all "connected" satisfying assignments of weight at most k, and constructs a solution by combining several of the assignments enumerated.

In this thesis, we investigate the enumeration aspects of two parameterized problems, both considered to be *facility location* problems: given a set of object and a distance function, we want to find an object that is as close as possible to the input set. In the first problem, NEIGHBOUR STRING, we have a bottleneck objective function: being close to a set of objects means being close to the farthest one. Problems with these types of objective functions are sometimes called centre problems. The objects in NEIGHBOUR STRING are strings. We formally define NEIGHBOUR STRING in Section 2.2.1. Aside from the application in coding theory [73, 78], to remove errors from sequences originating from a single sequence, this problem has several applications in computational biology, including drug target design [100], designing genetic probes to diagnose bacterial infections [100], finding regions of a DNA strand that are potential binding sites to a single transcription factor [126], designing universal PCR primers in order to amplify several regions of DNA simultaneously [100], and detecting an unbiased consensus sequence to represent a group of closely related sequences [15].

In the second problem, Kemeny Rank Aggregation, we have a cost function: being close to a set of objects means being close to the objects on average. Problems with this type of objective functions are sometimes called median problems. The objects in this problem are permutations (rankings of candidates). We formally define Kemeny Rank Aggregation in Section 2.3.1. The problem dates back to the 18th century [24, 42], when it was raised in the context of fair voting protocols, in France. For more context on applications in social choice, the reader is referred to the work of Kemeny and Snell[93], Bartholdi et al. [13, 14], and Bartholdi and Orlin [12]. The problem has also applications in planning problems in artificial intelligence [58, 59], bioinformatics [87], graph drawing [22], and in the general case of combining various preference lists each based on a certain criterion. Examples include ranking players based on their rankings in different eras, ranking films based on viewer rankings and critics' rankings, or designing a meta search engine based on the rankings provided by several search engines.

Both these problems are simplifications of the complicated applications from which they originate. Therefore, an enumeration of solutions will be useful to find a solution that really fits the application. Efficient enumeration of solutions is also helpful in deriving new properties and algorithms for these two problems.

We present the first parameterized enumeration algorithm for Neighbour String, the problem of determining a *neighbour string*, where a neighbour string of n input strings, each of length ℓ, and n input *distance allotments* (non-negative integers) is a string that differs from input s_i in no more than d_i positions, where d_i is the distance allotment corresponding to s_i. The problem is NP-complete even when the d_i's are equal, which is know as Closest String. Informally, it means that Closest String is unlikely to be solvable in polynomial time. The formal definition of the NP-complete class of problems is out of the scope of this thesis and the reader is referred to the excellent book of Papadimitriou [118] for a fairly complete treatment of the subject.

Our new approach gives us the ability to tune the running time to optimize the algorithm for varying relative values of n and $d = \max_i d_i$. For strings over an alphabet Σ, we can choose a tuning constant λ to obtain an algorithm that runs in time $O(n\ell + (nd)^{f(\lambda)}(|\Sigma| - 1)^d 5^{\lceil(1+\lambda)d\rceil})$, where f is a function that decreases with increasing λ.

Our enumeration algorithm compares well to parameterized algorithms for finding a single solution for Closest String. When $\Sigma = \{0, 1\}$, the dependency on d in our

$O^*(5^{\lceil(1+\lambda)d\rceil})$ time bound is an asymptotic improvement over the previous best parameterized time bound of $O^*(6.7308^d)$ to find a single solution.

We also present the first parameterized enumeration algorithm for Kemeny Rank Aggregation, the problem of determining an *optimal aggregation*, a total order that is at minimum total τ*-distance* from the input multi-set of m total orders (*votes*) over a set of n alternatives (*candidates*); the τ-distance between two total orders, denoted by k_t, is the number of pairs of candidates ordered differently in the two orders. We give an $O(nm + 4^{\frac{k_t}{m}} \cdot n^2)$-time enumeration algorithm and obtain a $4^{\frac{k_t}{m}}$ upper bound on the number of optimal aggregations. We demonstrate that this bound is tight, by giving a family of instances with $4^{\frac{k_t}{m}}$ optimal aggregations.

Our algorithm relies on the notion of locally optimal aggregations [57], total orders whose total τ-distance from the votes do not decrease by any single swap of two candidates adjacent in the ordering. In fact, our algorithm enumerates all locally optimal aggregations, extending an algorithm by Dwork et al. [57] which finds a single locally optimal aggregation. As a consequence of our approach, we provide not only an upper bound on the number of optimal aggregations, but also the first parameterized bound, $4^{\frac{k_t}{m}}$, on the number of locally optimal aggregations, which is a superset.

We complement our results by showing that any algorithm of running time $O(f(d)n^c)$ that computes a single solution for Neighbour String can be used to produce K solutions in time $O(K\ell \cdot |\Sigma| \cdot f(d)n^c)$, and any algorithm of running time $O(f(\frac{k_t}{m})n^c)$ that computes a single solution for Kemeny Rank Aggregation can be used to produce K solutions in time $O(K \cdot f(\frac{k_t}{m})n^{c+2})$. In particular, based on an $O^*(2^{O(\sqrt{\frac{k_t}{m}})})$-time algorithm [91] to find a solution for Kemeny Rank Aggregation [91], this gives a subexponential-time algorithm to enumerate a subexponential number of solutions. This "partial enumeration" technique can be combined with a general branching algorithm by Lawler [101], developed in 1972, for computing the first K *best* solutions of a optimization problem, giving algorithms of running times $O(K \log K \cdot f(k)n^{O(1)})$ that compute K best solutions for many optimization problems. Although the approach is not new, it has gone unnoticed in the parameterized community. In fact, this improves two published results.

As a side note, the "partial enumeration" technique used to produce K solutions for Neighbour String and Kemeny Rank Aggregation works for certain k-subset problems, where every solution is a subset of size at most k of a domain set. We observe that the size of our enumeration search tree can get large only if the solutions, corresponding to the nodes of the search tree, include large Δ-systems, where a Δ-system in the terminology of Erdös and Rado [61] is a family of Δ sets all intersecting in a common subset and no other element is in more than one set. From this perspective, the Erdös-Rado Theorem characterizes k-subset problems that have $f(k)$ solutions, for some function f.

We also demonstrate partial kernels (polynomial-time reductions to instances with m votes but small numbers of candidates) for Kemeny Rank Aggregation. The first of these partial kernels, which reduces the number of candidates to $\frac{4k_t}{m}$, is *enumerative*, that is, there is a bijection between solutions to the original instance and solutions to the kernel. We show how to reduce the number of candidates to $(2+\epsilon)\frac{k_t}{m}$ at the expense of losing the bijection, an improvement over the $\frac{16}{3}(\frac{k_t}{m})$-candidate partial kernel of Betzler et al. [18].

Our kernelizations use limited information from Kemeny Rank Aggregation instances. Consequently, the kernelization works for Weighted Directed Feedback Arc Set (the problem of finding a minimum-weight subset of arcs, in a weighted directed graph, whose removal makes the graph acyclic) on complete digraphs whose arc-weights satisfy the triangle inequality and the probability constraint (i.e. the weights of the arcs (a, b) and (b, a) add up to one for every pair of vertices a, b). As a result, we obtain a $4k$ vertex kernel, also enumerative, for Weighted Directed Feedback Arc Set (WDFAS) restricted to these digraphs. We also prove that WDFAS admits a not necessarily enumerative $(2+\epsilon)k$ vertex kernel, for any constant $\epsilon > 0$, for these digraphs; again, a kernel is a smaller instance of the same problem which has the same answer as the original instance. This result can be seen as an extension of a $(2+\epsilon)k$ vertex kernel for Unweighted Directed Feedback Arc Set due to Bessy et al. [16] to a special real-weighted variant for which no kernel was previously known.

We think that our proposed enumeration algorithms as well as our algorithms to enumerate K solutions, which use arbitrary algorithms to find a single solution, have applications in finding ideal solutions in practical applications. In the case of Kemeny Rank Aggregation, our enumerative kernel can always be used as the first step, to reduce the size of the problem and make the search even more efficient.

Chapter 2

Preliminaries

In the first part of this chapter, we briefly review several types of computational problems and complexity classes. Since computational complexity is not the main focus of this thesis, we sketch only the definitions that are used throughout the thesis. For a detailed study of the topic, the reader is referred to the classical books of Papadimitriou [118] and Downey and Fellows [54].

The definitions in the first part will form the basis for the parameterized enumeration algorithms for the Neighbour String and Kemeny Rank Aggregation problems in chapters 4, 5, and 6.

The rest of this chapter contains the definitions of Neighbour String and Kemeny Rank Aggregation, as well as observations that will be used in subsequent chapters.

2.1 Computational Complexity

The computational complexity framework has been developed to study the amount of resources, usually time and space, required to solve various types of computational problems. Various complexity classes corresponding to each type of problem are intended to characterize various levels of resource requirements.

In each of the following sections, we mention a problem type and a few complexity classes corresponding to that type of problem.

2.1.1 Decision Problems

We represent *decision problems* as languages over a finite alphabet Σ. We define Σ^* as the class of strings over Σ. Unless mentioned otherwise, we assume $\Sigma = \{0, 1\}$. The decision problem represented by a language asks for a Yes/No output depending on whether the input string is in the language or not. An algorithm (equivalently, a Turing machine) is said to *decide* a decision problem if it computes the corresponding Yes/No output for every input string.

It is generally assumed that a decision problem is efficiently solvable if it can be decided by a polynomial-time deterministic Turing machine.

Definition 2.1. *The class P is the class of all decision problems that are decidable by a polynomial-time deterministic Turing machine.*

The big open question of computational complexity is whether the class P is equal to the class NP:

Definition 2.2. *The class NP is the class of all decision problems that are decidable by a polynomial-time non-deterministic Turing machine.*

Both classes P and NP are closed under Karp reductions:

Definition 2.3. *A* Karp reduction *of a decision problem L_1 to a decision problem L_2 is a polynomial-time computable function f such that $x \in L_1$ if and only if $f(x) \in L_2$.*

A problem Q is NP-hard if every problem in NP has a Karp reduction to Q. An NP-hard problem is NP-complete if it is in NP.

2.1.2 Search Problems

A decision problem often corresponds to deciding whether the input instance has a certain structure. In this thesis, we are interested in actually finding one, or even several of, such structures. The generalized problem that specifies the desirable structures in its definition and asks for such a structure, instead of the Yes/No output, is known as a *search problem* [77]:

Definition 2.4. *The* search problem *represented as a binary relation R is the problem that given an input x asks for a string y such that $(x, y) \in R$, if there exists one, and for $\perp$ otherwise.*

We use $R(x)$ to denote $\{y : (x, y) \in R\}$.

The *decision problem associated with a search problem* R is $L(R) = \{x : R(x) \neq \emptyset\}$.

Example 2.1. Consider the binary relation $R = \{(x, w) : |w| > 0 \text{ and } www \text{ is a substring of } x\}$. The search problem represented by R receives a string x as input and outputs a substring w of x if x has www as a substring, and $\perp$ if x does not have a www substring for any string w. The decision problem associated with this search problem receives an input string x and outputs Yes/No depending on whether the input string has a substring of the form www. The language representing this decision problem is $L(R) = \{x : \exists w \text{ s.t. } |w| > 0 \text{ and } www \text{ is a substring of } x\}$.

Similar to decision problems, being polynomial-time solvable is a common assumption for efficiently-solvable search problems. In addition, a search problem is not considered efficiently-solvable if the lengths of strings in $R(x)$ are not polynomially bounded:

Definition 2.5. *A function* $f : \Sigma^* \mapsto \mathbb{N}$ *is* polynomially bounded *if there exists a polynomial function* p *such that for any input* x, $f(x) \leq p(|x|)$.

Definition 2.6. *A search problem* R *is* polynomially bounded *if* $f(x) = \max\{|y| : y \in R(x)\}$ *is polynomially bounded.*

Definition 2.7. *The class* PF *(Polynomial-time Find [79]) is the class of all search problems* R *that are polynomially bounded and are solvable by a polynomial-time deterministic Turing machine with an additional output tape.*

Definition 2.8. *The class* PC *(Polynomial-time Check [79]) is the class of all search problems* R *such that they are polynomially bounded and there exists a polynomial-time computable function that decides whether* $(x, y) \in R$ *for every* x *and* y.

The prefix of a search problem R links $R \in \text{PC}$ and $R \in \text{PF}$:

Definition 2.9. *The* prefix *of a search problem* R *is the search problem* $\textsc{Prefix}(R) = \{(x, y') : (x, y) \in R \text{ and } y' \text{ is a prefix of } y\}$.

Observation 2.1. *[79] If* $\textsc{Prefix}(R) \in PC$ *for a search problem* $R \in PC$*, then* $R \in PF$.

Proof. Algorithm 1 solves R using arbitrary algorithms to check $\textsc{Prefix}(R)$ and R. Checking $\textsc{Prefix}(R)$ helps the algorithm to avoid spending time on branches that do not lead to a solution. $\square$

Algorithm 1: Find

```
  Require: Two strings $x$ and $p$;    /* initially called with $p = \lambda$ (the empty
           string) */
1 if $(x, p) \in R$ then return $p$;
2 if $(x, p) \notin \textsc{Prefix}(R)$ then return $\perp$;
3 $y \leftarrow$ Find$(x, p0)$;                          /* $\Sigma$ is assumed to be $\{0, 1\}$ */
4 if $y \neq \perp$ then return $y$;
5 else return Find$(x, p1)$;
```

2.1.3 Optimization Problems

Optimization problems characterize a class of problems in which the desirable structures are partly specified by a measure function:

An *optimization problem* $\mathcal{Q}$ is represented as a three-tuple $(\mathcal{I}, S, m)$, where $\mathcal{I}$ is the set of *instances* of $\mathcal{Q}$, $S(x)$ is the set of *feasible solutions* for $x \in \mathcal{I}$, and $m(x, y)$ is the value of the feasible solution $y \in S(x)$. We use $\min(x)$ to denote $\min\{m(x, y) : y \in S(x)\}$. A feasible solution y for an instance $x \in \mathcal{I}$ is an *optimum solution* if $m(x, y) = \min(x)$. We use $\mathrm{Opt}(x)$ to denote the set of optimum solutions for x.

Definition 2.10. *The search problem associated with an optimization problem* $\mathcal{Q} = (\mathcal{I}, S, m)$ *is* $\textsc{Search}(\mathcal{Q}) = \{(x, y) : x \in \mathcal{I}, y \in \mathit{Opt}(x)\}$.

Definition 2.11. *The* prefix *of an optimization problem* $Q = (\mathcal{I}, S, m)$ *is the optimization problem* $\textsc{Prefix}(Q) = (\mathcal{I}', S', m')$*, where* $\mathcal{I}' = \{(x, p) : x \in \mathcal{I}\}$*,* $S'((x, p)) = \{y : y \in S(x)$ *and* p *is a prefix of* $y\}$*,* $m'((x, p), y) = m(x, y)$.

2.1.4 Enumeration Problems

Enumeration problems formulate the problem of finding all desirable structures in a search problem.

Definition 2.12. *The* enumeration *problem associated with a search problem* R *is the new search problem* $\textsc{Enum}(R) = \{(x, R(x))\}$.

We refer to an algorithm for $\textsc{Enum}(R)$ as an *enumeration algorithm* for R. Notice that $\textsc{Enum}(R)$ is a function. We have defined an enumeration problem as a search problem though, in order to keep uniformity.

Here, being polynomially bounded means to have polynomially many solutions in R (and R being polynomially bounded). Enumeration problems in PF are associated with search problems in PF whose sets of all solutions are polynomial-time computable. For enumeration problems that are not efficiently solvable (are not in PF) because of the large number of solutions, Valiant [130] suggested the use of the following class, which does not restrict the number of solutions yet limits the amount of time each solution takes on average to be produced.

Definition 2.13. *The class* P-enumerable *is the class of all enumeration problems Q that can be solved in $|Y| \cdot |x|^{O(1)}$ time for an input x such that $(x, Y) \in Q$.*

Examples of P-enumerable problems are enumeration of all perfect matchings in bipartite graphs [76] and enumeration of all vertices and faces of a convex polyhedron [75].

Observation 2.2. *[130] If* $\textsc{Prefix}(R) \in$ *PC for a search problem* $R \in$ *PC, then* $\textsc{Enum}(R) \in$ *P-enumerable.*

Proof. Algorithm 2 solves R using arbitrary algorithms to check $\textsc{Prefix}(R)$ and R. Checking $\textsc{Prefix}(R)$ helps the algorithm to avoid spending time on branches that do not lead to a solution. On the other hand, since the prefixes produced are all distinct, the solutions produced at different branches will be distinct; therefore, new solutions are always found after polynomial number of steps. □

Algorithm 2: Enumerate

```
Require: Two strings $x$ and $p$ ;   /* initially called with $p = \lambda$ (the empty string) */
1 if $(x,p) \notin \textsc{Prefix}(R)$ then return $\emptyset$;
2 $O \leftarrow$ Enumerate$(x, p0) \cup$ Enumerate$(x, p1)$;   /* $\Sigma$ is assumed to be $\{0,1\}$ */
3 if $(x,p) \in R$ then return $O \cup \{p\}$; else return $O$;
```

Standard reductions for search problems focus on preserving one solution in the reduced instance. In the case of enumeration problems, the goal is to produce all solutions. Therefore, it is very common that a reduction from a search problem R_1 to another search problem R_2 is not a reduction from $\textsc{Enum}(R_1)$ to $\textsc{Enum}(R_2)$.

Computing the cardinality of $R(x)$ is a relaxation of enumeration problems. The clean definition of counting problems, compared to enumeration problems, has made them the subject of complexity study, starting from Valiant's formalization of complexity classes for counting problems [129].

Definition 2.14. *The* counting *problem associated with a search problem R is the new search problem $\#R = \{(x, |R(x)|)\}$.*

There are problems in PF whose counting problems are hard to solve. An example of such problems, given by Valiant [129], is finding a false assignment for Boolean conjunctive normal form formulae, where an assignment making the input formula false can be found trivially, but counting the number of false assignments is at least as hard as recognizing if the formula is satisfiable, which is NP-hard.

Definition 2.15. *The class $\#P$ is the set of all counting problems associated with search problems in PC.*

The class $\#P$ is closed under parsimonious reductions, defined below:

Definition 2.16. *A* parsimonious reduction *of a counting problem R_1 to a counting problem R_2 is a polynomial-time computable function g such that $(x, y) \in R_1$ if and only if $(g(x), y) \in R_2$.*

A counting problem Q is *$\#P$-hard* if every counting problem in $\#P$ has a parsimonious reduction to Q. A $\#P$-hard problem is *$\#P$-complete* if it is in $\#P$.

For some NP-complete problems, the same reductions proving their NP-hardness prove that their corresponding counting problems are $\#P$-hard.

Searching for K solutions instead of all the solutions is another relaxation of enumeration problems:

Definition 2.17. *The* partial enumeration *problem associated with a search problem R is the search problem* $\textsc{Partial}(R) = \{((x, 1^K), y_1, y_2, \ldots, y_K) : \forall i\ (x, y_i) \in R$ *and the y_i's are distinct*$\}$.

We assumed that K is given in unary since otherwise $\textsc{Partial}(R)$ cannot be polynomially bounded unless $R(x)$ always has polynomial cardinality.

We refer to an algorithm for $\textsc{Partial}(R)$ as a *partial enumeration algorithm* for R.

In case of an optimization problem Q, $\textsc{Partial}(\textsc{Search}(Q))$ asks for a "partial enumeration" of optimal solutions. A broader definition which exploits the measure function also makes sense here:

Definition 2.18. *The* partial enumeration *problem associated with an optimization problem* $\mathcal{Q} = (\mathcal{I}, S, m)$ *is the search problem* $\text{PARTIAL}(\mathcal{Q}) = \{((x, 1^K), y_1, y_2 \ldots, y_K) : \forall i\ y_i \in S(x)$*, y_i's are distinct,* $m(x, y_1) \leq m(x, y_2) \leq \ldots \leq m(x, y_K)$*, and* $m(x, y_K) \leq m(x, y_{K+1})$ *for every* $y_{K+1} \in S(x) - \{y_1, \ldots, y_K\}\}$.

This definition is inspired by polynomial-time algorithms of Eppstein [60] and Chegireddy and Hamacheran [31] for finding K shortest paths and K best perfect matchings, respectively.

2.1.5 Parameterized Problems

In response to the inherent complexity of NP-hard problems, people have looked at approximate solutions of these problems or exact solutions for special input instances. The parameterized framework, introduced by Downey and Fellows [54], provides a tool for a systematic study of classes of inputs and the complexity of finding exact solutions for them. In this framework, the complexity of a problem is analyzed with respect to the lengths of input instances and a second "parameter" derived from the input.

Let us define a parameterized problem first.

Definition 2.19. *A* parameterized problem *is a decision problem* $L \subseteq \Sigma^* \times \mathbb{N}$*, for a finite alphabet* Σ.

In an input instance (x, k) to a parameterized problem, k is called the *parameter*. The parameters are typically assumed to be natural numbers.

We are interested in parameterized problems defined for hard-to-solve decision problems.

Definition 2.20. *A decision problem L* parameterized *by a function* $\kappa : \Sigma^* \mapsto \mathbb{N}$ *is the parameterized problem* $\{(x, \kappa(x)) : x \in L\}$.

One can see κ as a partitioning of Σ^* into $I_k = \{x \in \Sigma^* : \kappa(x) = k\}$ and L into $L_k = L \cap I_k$, referred as *slices* of input and L, respectively. For instance, in a graph problem parameterized by the maximum degree of vertices, input is partitioned into disjoint classes of graphs such that the kth class consists of graphs of maximum degree k. Intuitively, the goal behind such a partitioning of input instances is to study how much the parameter, the maximum degree in our example, captures the complexity of the problem.

Definition 2.21. *A* parameterized search problem *is the search problem $R \subseteq (\Sigma^* \times \mathbb{N}) \times \Sigma^*$, for a finite alphabet Σ.*

Definition 2.22. *A search problem R* parameterized *by a function $\kappa : \Sigma^* \mapsto \mathbb{N}$ is the parameterized search problem $\{((x, \kappa(x)), y) : (x, y) \in R\}$.*

So far, we have described the role of the parameter in partitioning the input and L. The art of the parameterized framework is to provide an efficiency measure that measures how well an algorithm decides each slice of L. The provided measure is tighter for slices that correspond to smaller values of the parameter.

Definition 2.23. *A function $r(x, k)$ is* fixed-parameter tractable *if $r(x, k) \leq f(k) \cdot |x|^{O(1)}$ for some function f. Note that f, here, may depend only on k and not on $|x|$.*

The class FPT, defined below, includes the parameterized problems assumed to be efficient in the parameterized framework. The choice of the parameter is an important factor. Indeed, parameterized study of a hard problem is not useful unless small values of the parameter cover many practical cases.

Definition 2.24. *The class* FPT *is the class of all parameterized problems that can be decided in fixed-parameter tractable time.*

We call an algorithm *fixed-parameter tractable (FPT)* if its running time for any input instance (x, k) is bounded by $r(x, k)$, for some fixed-parameter tractable function r. The O^* notion is sometimes used to eliminate polynomial factors of $|x|$ in fixed-parameter tractable functions.

Definition 2.25. *A function $r(x, k)$ is in $O^*(f(k))$ if $r(x, k) \in O(f(k) \cdot |x|^{O(1)})$.*

We must be careful with reductions of parameterized problems. The class FPT is not closed under a reduction that reduces the parameter to a new parameter that depends on the input length.

Definition 2.26. *A* parameterized reduction *of a parameterized problem L_1 to a parameterized problem L_2 is a pair of a polynomial-time function g and a fixed-parameter tractable function h such that $(x, k) \in L_1$ if and only if $(h(x, k), g(k)) \in L_2$.*

The main hierarchy of parameterized complexity classes is FPT $\subseteq W[1] \subseteq W[2] \subseteq \ldots$ XP, where W-hardness, shown using FPT reductions, is the analogue of NP-hardness in classical complexity. The complexity classes $W[1]$ and $W[2]$ are defined based on satisfiability problems for formulas in Conjunctive Normal Form; a formula is in *Conjunctive Normal Form (CNF)*, also called a *CNF formula*, if it is a conjunction of *clauses*, each of which is a disjunction of literals.

Weighted CNF-Sat

Input:	a CNF formula $\mathcal{F}$ and an integer k
Output:	a satisfying assignment for $\mathcal{F}$, which sets exactly k variables to true

A CNF formula is called a *c-CNF formula*, if each of its clauses consists of at most c literals. When the input formulas are restricted to 2-CNF formulas, Weighted CNF-Sat is called Weighted 2-CNF-Sat.

Definition 2.27. *The class* W[1] *is the class of all parameterized problems that can be reduced to* L(Weighted 2-CNF-Sat) *by a parameterized reduction.*

Definition 2.28. *The class* W[2] *is the class of all parameterized problems that can be reduced to* L(Weighted CNF-Sat) *by a parameterized reduction.*

Notice that Prefix(R), Enum(R), and $\#R$ are all parameterized search problems when R is a parameterized search problem. In order to have a parameterized search problem as Partial(R), we slightly misuse the notation:

Definition 2.29. *The* partial enumeration *problem associated with a parameterized search problem R is the parameterized search problem* Partial(R) $= \{(((x, 1^K), k), y_1, y_2 \ldots, y_K) : \forall i \ ((x, k), y_i) \in R$ *and the y_i's are distinct*$\}$.

We also need a new parsimonious reduction for parameterized counting problems:

Definition 2.30. *A* parameterized parsimonious reduction *of a parameterized counting problem R_1 to a parameterized counting problem R_2 is a pair of a polynomial-time function g and a fixed-parameter tractable function h such that $((x, k), y) \in R_1$ if and only if $((h(x, k), g(k)), y) \in R_2$.*

Definition 2.31. *The class* #W[1] *is the class of all parameterized counting problems that can be reduced to* #Weighted 2-CNF-Sat *by a parameterized parsimonious reduction.*

Definition 2.32. *The class* #W[2] *is the class of all parameterized counting problems that can be reduced to* #Weighted CNF-Sat *by a parameterized parsimonious reduction.*

A problem Q is $\#W[t]$*-hard* if for every problem P in $\#W[t]$ there exists a parameterized parsimonious reduction of P to Q. A $\#W[t]$-hard problem is $\#W[t]$*-complete* if it is in $\#W[t]$.

Parameterized Intractability

The assumptions made for parameterized intractability are weaker than P≠NP. Still, there is enough evidence, from the classical complexity, to believe them to be true. Below, we list the assumptions typically made; for more details, and for terminology, the reader is referred to the book by Niedermeier [113]:

FPT ≠ M[1]. The complexity class M[1] is the class of all parameterized problems that are parameterized-reducible to Mini-3-CNF-Sat, which is a modification of 3-CNF-Sat that receives two integers k and n in unary in addition to its 3-CNF formula F, and the number of variables in F is at most $k \log n$. Based on a result by Cai and Juedes [28], FPT=M[1] if and only if there exists an $O(2^{o(n)} \cdot |F|^{O(1)})$-time algorithm to evaluate the satisfiability of any n-variable 3-CNF formula F. It is believed to be very unlikely that such an algorithm exists [85]; this is known as *the Exponential-Time Hypothesis.* In the same paper, Cai and Juedes also show that, assuming FPT≠M[1], there is no $O(2^{o(k)} n^{O(1)})$-time algorithm for Vertex Cover and no $O(2^{o(\sqrt{k})} n^{O(1)})$-time algorithm for Planar Vertex Cover [28].

FPT ≠ W[1]. The complexity class $W[1]$ is the first class in the Weft hierarchy, believed to specify computationally-different classes of parameterized problems. Equivalence of FPT and W[1] would result in FPT=M[1] (since FPT $\subseteq$ M[1] $\subseteq$ W[1]), and thus is very unlikely. The reverse is still not known to be true and so, currently, FPT≠M[1] is considered a stronger assumption than FPT≠W[1]. As an example result, Partial Vertex Cover (a generalization of Vertex Cover which asks whether there exists a subset of k vertices that covers at least t edges for any given integers k and t) is unlikely to have an FPT algorithm with respect to the parameter k [113].

Various Parameterizations

The following categories are commonly used in the literature for parameterizations of Search($\mathcal{Q}$) for an optimization problem $\mathcal{Q}$.

Standard parameterization. The parameter function $\min(x)$ is a standard choice of the parameter, perhaps because $\min(x)$ reflects the complexity of solving the problem for an instance x in many cases.

Structural parameterization. Other choices for the parameter are *structural* measures of the input instance, as opposed to the standard parameter which is considered more of an output-driven parameter. Of course, even the output totally depends on the input, and an input-driven/output-driven parameter refers to a parameter immediately seen in the input/output.

Above-guarantee parameterization. Mahajan et al. [106, 107] suggested modifying a parameterization if all non-trivial instances correspond to large values of the parameter. They argued that the parameter will not catch the complexity of the problem without this change. In the extreme case that L_k is trivially decidable for all k's smaller than $g(|x|)$ for a function g, k will depend on $|x|$ for all non-trivial instances, thus a fixed-parameter algorithm can spend an arbitrary amount of time to solve non-trivial instances. A more meaningful parameterization will be parameterizing the problem with the new parameter $k - g(|x|)$.

Example 2.2. Consider the optimization problem $\mathcal{Q}$ = Minimum Vertex Cover. The standard parameter for Search($\mathcal{Q}$) is the size of the minimum vertex cover in graph instances. In this example, the tree-width of the input graph is a structural parameter. On the other hand, consider a fake assumption that any vertex cover is guaranteed to include at least half the graph vertices. Then, the size of the minimum vertex cover minus half the number of vertices would be an above-guarantee parameter.

Parameterized Techniques

Over the years, a tool box for development of FPT algorithms has been created. Among the various techniques in the tool box, we will only use bounded search trees and kernelizations in this thesis. Nevertheless, we give an overview of the techniques that will be mentioned in the previous work, in order for the reader to compare our algorithms with other paradigms used in the literature. Detailed descriptions of the techniques along with examples can be found in the book by Niedermeier [113] and in the more recent surveys of Sloper and Telle [124] and Downey and Thilikos [55].

Bounded search tree. A branching algorithm that is recursive and the height of its execution tree is a function of k. Typically, this algorithm spends polynomial time in each node of its execution tree. Not every number of branches in the nodes makes the algorithm fixed-parameter tractable, but an upper bound of $O(\log(n)^{g(k)} \cdot h(k))$ on the number of branches, for arbitrary functions g and h, suffices to ensure fixed-parameter tractability [124].

Integer linear programming. Any integer program of length n having p variables can be solved in $O(p^{9p/2}n)$ arithmetic operations over integers of $O(p^{2p}n)$ bits. [102, 89, 99]. As a consequence, a parameterized problem is in FPT if it can be formulated by an integer program whose number of variables depends solely on the parameter.

Color-coding. This technique was originally developed to check whether an input graph has a path or cycle of length k [8]. Nevertheless, it applies to many search problems that look for small structures in the input graph [8]. The idea is to use a family of $f(k)$ coloring functions (families of perfect hash functions) that covers all k-subsets of an n element ground set, meaning that the members of any k-subset of the ground set are coloured distinctly by at least one of the functions. Therefore, an algorithm can go through all the coloring functions and solve a "colored" version of the problem, for each coloring, where the solution is assumed to be distinctly colored. Usually, the "colored" version is solved using dynamic programming (or divide and conquer, at a slightly worst running time and a polynomial space complexity instead of the exponential space complexity in the dynamic programming approach).

Kernelization. Kernelization is the method of reducing input instances to *kernels*, instances of size depending only on a (small) parameter k; in this respect it is the preprocessing phase performed in algorithms used in practice, and is hence a topic of wide interest.

Reduction to a kernel is often an iterative process based on the exhaustive application of a set of *reduction rules.* Combined with such a polynomial-time reduction, any algorithm that solves the problem for kernels, even through brute-force, gives a fixed-parameter algorithm for the original problem. As a matter of fact, a problem reduces to a kernel if and only if it has a fixed-parameter algorithm [54].

There has been interest in reducing, in polynomial time, a "dimension" of input instances to a function of the parameter. In a matrix problem, this could be reducing the number of rows to $f(k)$. Due to the similarities to kernels, the reduced instances are called *partial kernels* [18] in this case.

Tree-width. Many graph problems are in FPT when parameterized by the tree-width of input instances. There is a trend, initiated by Courcelle's Theorem [45], of characterizing such graph problems. For instance, it is been shown that any graph problem that can be stated in monadic second order logic is FPT with respect to the tree-width parameter. This technique becomes more interesting when a graph problem is parameterized by another parameter that turns out to be a function of tree-width. For example, any planar graph that has a k-dominating set has treewidth $O(\sqrt{k})$ [6]. In consequence, PLANAR k-DOMINATING SET is proved to be in FPT.

2.2 Neighbour String

In the well-studied CLOSEST STRING problem [73, 126, 15, 103, 78, 82, 38], and its generalization NEIGHBOUR STRING [105], the goal is to determine a string that is not too different from any of the n length-ℓ input strings. In the former case, the solution cannot differ from any string s_i in more than d positions; in the latter case, for each s_i a bound d_i is specified as part of the input, and the solution cannot differ from s_i in more than d_i positions.

2.2.1 Definitions

Throughout this thesis, we use Σ^ℓ to denote the set of all strings of length ℓ over the alphabet Σ. For any $s \in \Sigma^\ell$, the character in position p in s, $1 \leq p \leq \ell$, is denoted by $s[p]$. For a set of positions or *region* $R \subseteq \{1, 2, \ldots, \ell\}$, we use $\overline{R}$ to denote $\{1, 2, \ldots, \ell\} - R$, the positions not in the region R. We define $s|R$ as the string formed by removing the characters in positions in $\overline{R}$, and making the remaining characters consecutive. For the example $s = 01101$ and $R = \{1, 3, 4\}$, $s|R$ is the string 010. By extension, for a set of strings $S \subseteq \Sigma^\ell$, $S|R$ is defined as $\{s|R : s \in S\}$. For convenience, we call $s|R$ (respectively, $S|R$) the *restriction of s (respectively, S) to R*. For strings s_1 and s_2 of lengths $|R|$ and $\ell - |R|$, we define $s_1 \oplus_R s_2$ as the string s of length ℓ such that $s|R = s_1$ and $s|\overline{R} = s_2$, and $s_1 \oplus_R S$ as the set of strings $\{s_1 \oplus_R s_2 : s_2 \in S\}$.

We use a distance measure to describe the degree to which strings differ from each other. For two strings $s_1, s_2 \in \Sigma^\ell$, $P(s_1, s_2)$ is defined as the set $\{p : s_1[p] \neq s_2[p]\}$ of positions on which the strings differ. The *Hamming distance* between s_1 and s_2, denoted by $H(s_1, s_2)$, is the number of positions in $P(s_1, s_2)$; more informally, we say that s_1 is *at distance $H(s_1, s_2)$ from s_2*. For any $d \geq H(s_1, s_2)$, we say that s_1 (respectively, s_2) is *within distance d* of s_2 (respectively, s_1).

We are now ready to define the problem of NEIGHBOUR STRING, in which each solution must be close to all given input strings, associated with a non-negative integer (the threshold distances supplied). NEIGHBOUR STRING is the search problem defined as the set of the following (input, output) pairs:

NEIGHBOUR STRING

Input:	a multiset $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n)\}$, where $s_i \in \Sigma^\ell$ and d_i is a non-negative integer (the *distance allotment*) for $1 \le i \le n$
Output:	a string $\sigma \in \Sigma^\ell$ such that for all i, $H(\sigma, s_i) \le d_i$

We call σ a *neighbour string* of $\mathcal{I}$. We use $\mathcal{I}_s$ to denote the set of strings in instance $\mathcal{I}$, that is, $\mathcal{I}_s = \{s_1, \ldots, s_n\}$, and $\text{NS}(\mathcal{I})$ to denote the set of all neighbour strings for $\mathcal{I}$.

Example 2.3. Consider the instance $\mathcal{I} = \{(00000, 3), (11110, 3), (00111, 2)\}$ of NEIGHBOUR STRING. The set of strings in this instance, i.e. $\{00000, 11110, 00111\}$, is denoted by $\mathcal{I}_s$. The string 00111 is a neighbour strings for $\mathcal{I}$, and therefore $00111 \in \text{NS}(\mathcal{I})$.

NEIGHBOUR STRING was originally defined as a generalization of CLOSEST STRING [105], another search problem defined as the set of the following (input, output) pairs:

CLOSEST STRING

Input:	a multiset $\mathcal{I} = \{(s_1, d), (s_2, d), \ldots, (s_n, d)\}$, where $s_i \in \Sigma^\ell$ and d is a non-negative integer (the *distance allotment*)
Output:	a string $\sigma \in \Sigma^\ell$ such that for all i, $H(\sigma, s_i) \le d$

We call σ a *closest string* of $\mathcal{I}$. We use $\mathcal{I}_s$ to denote the set of strings $\{s_1, \ldots, s_n\}$.

In this thesis, we study the search problem associated with an optimization version of NEIGHBOUR STRING, defined as the set of the following (input, output) pairs:

OPTIMIZATION NEIGHBOUR STRING

Input:	a multiset $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n)\}$, where $s_i \in \Sigma^\ell$ and d_i is a non-negative integer for $1 \le i \le n$
Output:	a string $\sigma \in \Sigma^\ell$ that minimizes d such that $H(\sigma, s_i) \le d_i - (\max_i d_i - d)$, for all i

Example 2.4. Consider the previous instance $\mathcal{I} = \{(00000, 3), (11110, 3), (00111, 2)\}$ of Neighbour String, which is also an instance of Optimization Neighbour String. The minimum d in this example is 2, meaning that the maximum d_i can be reduced to 2 and all other d_i's can be decreased by the same amount, yet the resulting instance has a neighbour string. That is, the instance $\{(00000, 2), (11110, 2), (00111, 1)\}$ has a neighbour string 00110, and this is the largest distance decrease we can have: the instance $\{(00000, 1), (11110, 1), (00111, 0)\}$ has no neighbour string. The solutions to Optimization Neighbour String are the set of neighbour strings of $\{(00000, 2), (11110, 2), (00111, 1)\}$. Therefore, 00111 is not considered a solution, though satisfying the original distance allocations, but 00110 is.

Optimization Neighbour String is a generalization of Optimization Closest String, defined as the set of the following (input, output) pairs:

Optimization Closest String

Input:	a multiset $\mathcal{I} = \{(s_1, 0), (s_2, 0), \ldots, (s_n, 0)\}$, where $s_i \in \Sigma^\ell$
Output:	a string $\sigma \in \Sigma^\ell$ that minimizes d such that for all i, $H(\sigma, s_i) \leq d$

We consider the standard parameterization for both Optimization Neighbour String and Optimization Closest String problems; that is, d will be the parameter. We use P-Neighbour String and P-Closest String to refer to these two problems, respectively.

We assume that we know d. Otherwise, we can always start solving the problem with $d = 0$. If no solution is found, we resolve the problem with $d = 1$, and so on. By the time d is set to its correct value (the minimal d for which a solution exists), a solution will be found. The whole process takes at most d times (or $\log d$ times, if binary search is used) the running time required to solve the problem with the extra assumption that d is given as part of the input.

In the case of the Optimization Neighbour String problem, we also assume that $d = \max_i d_i$. Otherwise, we can always adjust the d_i values to $d_i - (\max_i d_i - d)$.

Having both these assumptions, P-Neighbour String (P-Closest String, respectively) can be viewed as the restriction of Neighbour String (Closest String, respectively), when parameterized by $\max_i d_i$ (d, respectively), to minimal instances:

Definition 2.33. *An instance $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n)\}$ is* minimal *if for no string s, $H(s, s_i) \leq d_i - 1$ for all i.*

P-Neighbour String

Input:	a minimal instance $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n)\}$, where $s_i \in \Sigma^\ell$ and d_i is a non-negative integer (the *distance allotment*) for $1 \le i \le n$
Output:	a string $\sigma \in \Sigma^\ell$ such that for all i, $H(\sigma, s_i) \le d_i$

P-Closest String

Input:	a minimal instance $\mathcal{I} = \{(s_1, d), (s_2, d), \ldots, (s_n, d)\}$, where $s_i \in \Sigma^\ell$ and d is a non-negative integer (the *distance allotment*)
Output:	a string $\sigma \in \Sigma^\ell$ such that for all i, $H(\sigma, s_i) \le d$

2.2.2 Key Properties

When there are two input strings s_1 and s_2, Neighbour String has a solution if and only if $|P(s_1, s_2)| \le d_1 + d_2$ (i.e. the number of positions in which s_1 and s_2 differ is at most $d_1 + d_2$), a result of the following general observation:

Observation 2.3. *Suppose that s_x and s_y are arbitrary strings and $R \subseteq P(s_x, s_y)$ for a region R. Then, for any string π, $H(\pi|R, s_x|R) + H(\pi|R, s_y|R) \ge |R|$.*

Proof. Since R is a region in which s_x and s_y disagree, π and s_y disagree in at least those positions in R in which π and s_x agree. Therefore, $H(\pi|R, s_y|R) \ge |R| - H(\pi|R, s_x|R)$, as required. □

More can be said for $R = P(s_x, s_y)$: since $H(\pi|\overline{R}, s_x|\overline{R})$ equals $H(\pi|\overline{R}, s_y|\overline{R})$, Observation 2.3 results in the following observation:

Observation 2.4. *Suppose that $H(\pi, s_x) \le d_x$ and $H(\pi, s_y) \le d_y$ for arbitrary strings s_x, s_y, and π. Then, assuming that $R = P(s_x, s_y)$, $H(\pi|\overline{R}, s_x|\overline{R}) = H(\pi|\overline{R}, s_y|\overline{R}) \le \frac{d_x + d_y - |R|}{2}$.*

Proof. Considering distances in R and $\overline{R}$, we have

$$(H(\pi|R, s_x|R) + H(\pi|\overline{R}, s_x|\overline{R})) + (H(\pi|R, s_y|R) + H(\pi|\overline{R}, s_y|\overline{R})) \le d_x + d_y$$

Consequently, $H(\pi|\overline{R}, s_x|\overline{R}) + H(\pi|\overline{R}, s_y|\overline{R})$ is less than or equal to $(d_x + d_y - H(\pi|R, s_x|R) - H(\pi|R, s_y|R)$, which is at most $d_x + d_y - |R|$ by Observation 2.3.

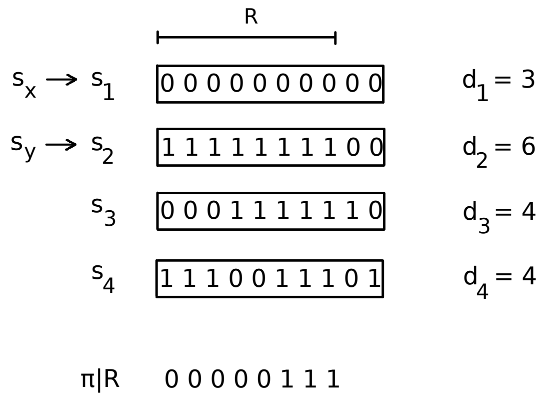

Figure 2.1: Example strings and distances restricted to a region $R = P(s_x, s_y)$

Since R is the region in which s_x and s_y disagree, $H(\pi|\overline{R}, s_x|\overline{R}) = H(\pi|\overline{R}, s_y|\overline{R})$, and therefore, they both are at most $\frac{d_x+d_y-|R|}{2}$, as required. $\square$

Example 2.5. Consider the instance $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), (s_3, d_3), (s_4, d_4)\}$ demonstrated in Fig. 2.1, and a string π with $\pi|R = 00000111$. In each of the positions in $R = P(s_1, s_2)$, π is either different from s_1 or s_2. In fact, $H(\pi|R, s_1|R) + H(\pi|R, s_2|R) = |R|$. If $H(\pi, s_1) \leq d_1 = 3$ and $H(\pi, s_2) \leq d_2 = 6$, then $H(\pi|\overline{R}, s_x|\overline{R}) = H(\pi|\overline{R}, s_y|\overline{R}) \leq \frac{d_x+d_y-|R|}{2} = \frac{9-8}{2}$. Consequently, $\pi|\overline{R}$ has to be 00.

When $|R| \geq d_y$, Observation 2.4 simplifies to $H(\pi|\overline{R}, s_x|\overline{R}) \leq \frac{d_x}{2}$. As a consequence, most of the positions that differ between a neighbour string $\sigma \in \text{NS}(\mathcal{I})$ and an input string $s_x \in \mathcal{I}_s$ are located in $R = P(s_x, s_y)$ for a "far" input string $s_y \in \mathcal{I}_s$. Indeed, if an algorithm somehow determines the restriction of σ to R, all it needs is to find the at most $\frac{d_x}{2}$ positions that differ between $\pi|\overline{R}$ and $s_x|\overline{R}$.

Such a "far" input string $s_y \in \mathcal{I}_s$ (of distance at least d_y from s_x) always exists if $s_x \notin \text{NS}(\mathcal{I})$:

Observation 2.5. *If $s_x \notin \text{NS}(\mathcal{I})$, then there exists $s_y \in \mathcal{I}$ such that $|P(s_x, s_y)| > d_y$.*

These observations were originally used by Ma and Sun [105] giving the first algorithms for Closest String and Neighbour String whose time bounds had exponential dependence on d. We have changed the statements of the observations to fit our discussions in Chapter 4.

We observe that the condition $s_x \notin \text{NS}(\mathcal{I})$ is not needed for a "far" input string s_y to exist when $\mathcal{I}$ is a minimal instance. A "far" $s_y \in \mathcal{I}_s$ always exists in a minimal instance since otherwise s_x is a string at distance smaller than d_i from s_i, for all s_i's, which is not possible for such an instance. We will use this property to continue the search for neighbour strings even if one of the input strings is a neighbour string already. As a consequence, we will be able to enumerate all neighbour strings of minimal instances.

2.3 Kemeny Rank Aggregation

Preference lists are typical elements of elections, psychology questionnaires, and every-day surveys. Social choice theory studies the problem of combining the gathered individual preference lists into a consensus preference list that reflects the expressed preferences as closely as possible. The problem of finding a single preference list that on average is as close as possible to the given preference lists, according to a chosen distance measure, is called Rank Aggregation. In this thesis, we study Kemeny Rank Aggregation, introduced by Kemeny in 1959 [92], a famous version of Rank Aggregation, where the distance between two preference lists is the number of pairs of candidates that are preferred differently in the two lists.

2.3.1 Definitions

Complete or partial preference lists over a set of candidates U can be represented as binary relations, namely sets of ordered pairs in $U \times U$, where each ordered pair (x, y) in the relation represents the preference of a candidate x to a candidate y. As a benefit, set operations can be used to manipulate preference lists. For instance, the number of common preferences in two preference lists π_1 and π_2 can be represented as $\pi_1 \cap \pi_2$. For consistency, we also treat graph arcs as ordered pairs and sets of arcs as binary relations that consist of the corresponding ordered pairs. At times we restrict a binary relation R to a set of unordered pairs P, where $R|P = \{(x, y) \in R : \{x, y\} \in P\}$; similarly, for E a set of ordered pairs (i.e. a binary relation), $R|E = \{(x, y) \in R : (x, y) \in E \text{ or } (y, x) \in E\}$.

We use $x <_R y$ to denote $(x, y) \in R$, for a binary relation $R \subseteq U \times U$, to show the preference of x to y. The *endpoints* of an ordered pair (x, y) are x and y, where x is the *head* and y is the *tail*; we use endpoints(R) for a binary relation to denote $\bigcup_{e \in R}$ endpoints(e). The *reverse* of an ordered pair (x, y), denoted by rev*(R)*, is the ordered pair of reversed head and tail (i.e., (y, x)). The *reverse* of a binary relation, representing the preferences opposite to those in R, denoted by rev*(R)*, is defined as $\{(y, x) : (x, y) \in R\}$.

In many cases, preference lists are acyclic:

Definition 2.34. *A binary relation R is* transitive *if $z <_R x$ and $x <_R y$ imply $z <_R y$.*

We use R^+ for a binary relation R to denote the binary relation of minimum cardinality that is a superset of R and is transitive, often called the transitive closure of R.

Definition 2.35. *A binary relation R is* acyclic *if $R \cap \mathrm{rev}(R^+) = \emptyset$.*

We assume further that preference lists are total orders, and the consensus preference list is also a total order; other variants of the problem are briefly mentioned in Chapter 3.

Definition 2.36. *A binary relation R is a* total order *over a set U if it is transitive, for any $x <_R y$, x is not equal to y and $y \not<_R x$, and for any $x, y \in U$, $x \neq y$, either $x <_R y$ or $y <_R x$.*

We use *Total*(U) to denote the set of total orders over U.

We use a distance measure to describe the degree to which total orders differ from each other.

Definition 2.37. *The* τ-distance *between $\pi_1 \in$ Total(U) and $\pi_2 \in$ Total(U), denoted by $\tau(\pi_1, \pi_2)$, is the number of pairs in $\pi_1 \setminus \pi_2$, and by extension, the* τ-distance *between π_1 and a multi-set $\mathcal{I}$ over Total(U), denoted by $\tau(\pi_1, \mathcal{I})$, is $\sum_{\pi_2 \in \mathcal{I}} \tau(\pi_1, \pi_2)$.*

Since the numbers of pairs in any $\pi_1 \in \mathrm{Total}(U)$ and $\pi_2 \in \mathrm{Total}(U)$ are the same, $\tau(\pi_1, \pi_2) = \tau(\pi_2, \pi_1)$.

We are now ready to define the problem of Kemeny Rank Aggregation, in which the input is a multi-set of total orders and each solution must be close to the input multi-set. Kemeny Rank Aggregation is the search problem defined as the set of the following (input, output) pairs:

Kemeny Rank Aggregation

Input:	a multi-set $\mathcal{I}$ of m total orders in $\mathrm{Total}(U)$ where U is a set of n elements
Output:	a total order $\sigma \in \mathrm{Total}(U)$ that minimizes $k_t = \tau(\sigma, \mathcal{I})$

The measure k_t is the *total* τ-distance between σ and the total orders in $\mathcal{I}$, versus k_a or k_m used to denote average and maximum τ-distances [123].

We refer to the elements in U as *candidates* and to the total orders in $\mathcal{I}$ as *votes*. We also call any total order that minimizes $k_t = \tau(\sigma, \mathcal{I})$ an *optimal aggregation* of $\mathcal{I}$, and use $\mathrm{KRA}(\mathcal{I})$ to denote the set of all optimal aggregations of $\mathcal{I}$.

In this thesis, we study the parameterization of KEMENY RANK AGGREGATION by the parameter $\frac{k_t}{m}$, which is a normalization of the standard parameter k_t. The reason we are not using k_t itself is that its value is generally large for instances with large m, and thus small values of k_t do not cover many practical instances. We use P-KEMENY RANK AGGREGATION to refer to this parameterized search problem.

2.3.2 Preferences that are Agreed on

None of the optimal aggregations can disagree with an all-agreed-on preference, since a disagreement with any such preference can only increase the τ-distance from $\mathcal{I}$. We let *unanimity*$(\mathcal{I})$ denote the binary relation $\bigcap_{\pi\in\mathcal{I}}\pi$. Preferences in *unanimity*$(\mathcal{I})$ are preferences that are common in all total orders in $\mathcal{I}$.

Observation 2.6. *[110] For any $\sigma \in \mathrm{KRA}(\mathcal{I})$, unanimity$(\mathcal{I}) \subseteq \sigma$.*

Therefore, P-KEMENY RANK AGGREGATION reduces to determining the order of dirty pairs, defined below:

Definition 2.38. *The set of* dirty pairs *of $\mathcal{I}$, denoted by* $\mathrm{dirty}(\mathcal{I})$, *is* $\{\{a,b\} : (a,b) \in \bigcup_{\pi\in\mathcal{I}}(\pi \setminus \mathit{unanimity}(\mathcal{I}))\}$.

2.3.3 Connection to Feedback Arc Set

We use a well-known reduction to WEIGHTED DIRECTED FEEDBACK ARC SET (WDFAS) on complete digraphs [94], where a *feedback arc set* F for a graph G is a subset of the graph arcs whose removal makes the graph acyclic, with *weight* $w_F = \sum_{(a,b)\in F} w(a,b)$.

WDFAS

Input:	an arc-weighted directed graph G
Output:	a feedback arc set F for G of minimum weight

We use $\mathrm{MF}(V,w)$ to denote the set of all minimum feedback arc sets in a complete digraph G on the vertex set V and with the arc-weight function w. Feedback arc sets in a complete digraph must have many arcs; each must include a total order: removing the arcs of a feedback arc set F must render the remaining graph acyclic, and thus consistent with at least one total order H (a topological order of the remaining graph), proving that the total order $\mathrm{rev}(H)$ was included in the removed arcs, i.e., $\mathrm{rev}(H) \subseteq F$. Therefore, since each total order in $\mathrm{Total}(V)$ is a feedback arc set for G, the total orders in $\mathrm{Total}(V)$ are exactly the *minimal* feedback arc sets (sets for which the removal of any arc will result in a cycle in the remaining graph); thus since every minimum weight feedback arc set is minimal, $\mathrm{MF}(V, w) \subseteq \mathrm{Total}(V)$.

An instance $\mathcal{I}$ of P-Kemeny Rank Aggregation is reduced to a complete digraph with arc-weights between zero and one:

Observation 2.7. *A total order $\sigma \in \mathit{Total}(U)$ is an optimal aggregation of $\mathcal{I}$ if and only if $\mathrm{rev}(\sigma) \in \mathrm{MF}(U, w)$, where $\mathcal{I}_{(a,b)}$ is $\{\pi \in \mathcal{I} : a <_\pi b\}$ and w is the weight function $w(a, b) = \frac{|\mathcal{I}_{(a,b)}|}{m}$. Also, $\sum_{(a,b)\in\mathrm{rev}(\sigma)} w(a, b) = \frac{k_t}{m}$, where $k_t = \tau(\sigma, \mathcal{I})$.*

Proof. The τ-distance between any total order $\pi \in \mathrm{Total}(U)$ and $\mathcal{I}$ is precisely m times the weight of $\mathrm{rev}(\pi)$ in the complete digraph with vertex set U and the arc-weight function $w(a, b) = \frac{|\mathcal{I}_{(a,b)}|}{m}$.

Since every optimal aggregation of $\mathcal{I}$ and every feedback arc set in $\mathrm{MF}(V, w)$ is in $\mathrm{Total}(U)$, any $\sigma \in \mathrm{Total}(U)$ is an optimal aggregation of $\mathcal{I}$ if and only if $\mathrm{rev}(\sigma) \in \mathrm{MF}(U, w)$. $\square$

The weight function in the reduced instance satisfies two useful properties, which we will exploit in the analysis of our algorithm, presented in Chapter 5. A weight function $w : U \times U \mapsto R$ satisfies the *probability constraint* if $w(a, b) + w(b, a) = 1$ for all pairs $a, b \in U$.

Observation 2.8. *[133] The weight function $w(a, b) = \frac{|\mathcal{I}_{(a,b)}|}{m}$ satisfies the probability constraint and the triangle inequality.*

2.3.4 The Impact of Large Weights

Relative weights of arcs influence the set of arcs included in optimal aggregations. In this section, we mention one example of the impact of heavily-weighted arcs. We will see more such examples in Section 3.4.3.

Let us first define classes of arcs based on the weights of the arcs. For a weight function $w : V \times V \mapsto R$, we define the binary relations $w_{<c}$, $w_{=c}$, and $w_{\neq c}$ as $\{(a, b) : w(a, b) < c\}$, $\{(a, b) : w(a, b) = c\}$, and $\{(a, b) : w(a, b) \neq c\}$, respectively; $w_{\leq c}$, $w_{>c}$ and $w_{\geq c}$ are defined analogously.

We are looking for properties of vertices that may be adjacent in minimum feedback arc sets:

Definition 2.39. *An ordered pair (x, y) is* π-adjacent *for a total order $\pi \in$ Total(V) if $x <_\pi y$ and there is no $z \in V$ such that $x <_\pi z <_\pi y$. We use* $\mathrm{adj}(\pi)$ *to denote the binary relation consisting of all π-adjacent ordered pairs.*

Two total orders that have the same set of adjacent ordered pairs are equal.

Example. Let $V = \{1, 2, 3, 4\}$ and $\sigma \in \mathrm{Total}(V)$ satisfy $1 <_\sigma 2 <_\sigma 3 <_\sigma 4$. Then, the set of σ-adjacent ordered pairs is $\mathrm{adj}(\sigma) = \{(1, 2), (2, 3), (3, 4)\}$.

For any $\sigma \in \mathrm{MF}(V, w)$, all σ-adjacent arcs are in $w_{\leq \frac{1}{2}}$, since otherwise replacing a violating arc with its reverse decreases the weight of σ, a contradiction:

Observation 2.9. *Suppose that $\sigma \in \mathrm{MF}(V, w)$. Then,* $\mathrm{adj}(\sigma) \subseteq w_{\leq \frac{1}{2}}$.

As a consequence, if for some $V' \subseteq V$ every arc in $w_{< \frac{1}{2}}$ between the vertices in V' and $V \setminus V'$ is directed from V' to $V \setminus V'$, then none of the arcs in $\mathrm{adj}(\sigma)$, for any $\sigma \in \mathrm{MF}(V, w)$, can be directed from $V \setminus V'$ to V':

Observation 2.10. *Every $F \in \mathrm{MF}(V, w)$ includes $(V' \times (V \setminus V'))$ for every subset V' of vertices V such that $(V' \times (V \setminus V')) \subseteq w_{< \frac{1}{2}}$.*

In the aggregation setting (consider the reverse of each arc), this is in fact in agreement with the Extended Condorcet Criterion [128].

Definition 2.40. *[128] A total order σ satisfies* the Extended Condorcet Criterion *if for every subset U' of candidates U such that $(U' \times (U \setminus U')) \subseteq w_{> \frac{m}{2}}$, where $w(a, b) = |\mathcal{I}_{(a,b)}|$, $(U' \times (U \setminus U')) \subseteq \sigma$.*

This criterion states that if a group of candidates (here, U') is pairwise preferred to the candidates not in the group (i.e. $U \setminus U'$) by majority of voters, a fair final result does not rank any of the candidates in the former group lower than any of the candidates in the latter group.

Observation 2.11. *[128] Every optimal aggregation satisfies the Extended Condorcet Criterion.*

2.3.5 Locally Optimal Solutions

In the Kemeny Rank Aggregation setting, Dwork et al. [57] defined local optimality for a total order as having the minimum cost among the total orders at τ-distance 1 from it. The incentive was to relax the conditions for an optimal aggregation to make it efficiently computable. We repeat the definition here since our fixed-parameter algorithm in Chapter 5 does not merely solve Enum(P-Kemeny Rank Aggregation); it enumerates all locally optimal total orders.

A closer look at total orders at τ-distance 1 from a total order gives rise to the following equivalent definition [57].

Definition 2.41. *A total order $\sigma \in Total(U)$ is a* locally optimal aggregation *for an instance $\mathcal{I}$ of m total orders of* Kemeny Rank Aggregation *if* $\mathrm{adj}(\sigma) \subseteq n_{\geq \frac{m}{2}}$ *for the weight function $n(a,b) = |\mathcal{I}_{(a,b)}|$.*

Naturally, every optimal aggregation is a locally optimal aggregation. In fact, the set of locally optimal aggregations is expected to be much larger than the set of optimal aggregations. Notice that the definition does not depend on the value of k_t. As a result, sets of locally optimal aggregations for P-Kemeny Rank Aggregation instances are equal to sets of locally optimal aggregations for the corresponding Kemeny Rank Aggregation instances.

We define locally minimum feedback arc sets in digraphs in analogy with locally optimal aggregations.

Definition 2.42. *A feedback arc set F is* locally minimum *if it is minimal and* $\mathrm{adj}(F) \subseteq w_{\leq \frac{1}{2}}$.

A minimal feedback arc set is a locally minimum feedback arc set if reversing a single arc does not produce a feedback arc set of smaller weight. We use $\mathrm{LF}(V, w)$ to denote the set of all locally minimum feedback arc sets in the complete digraph on the vertex set V and the arc-weight function w.

By the minimality condition, locally minimum feedback arc sets are forced to be total orders, making them comparable to locally optimal aggregations.

Observation 2.12. *A total order $\sigma \in Total(U)$ is a locally optimal aggregation for an instance $(\mathcal{I}, k)$ of* P-Kemeny Rank Aggregation, *where $\mathcal{I}$ is a multi-set of m total orders, if and only if* $\mathrm{rev}(\sigma) \in \mathrm{LF}(U, w)$, *for the weight function* $w(a,b) = \frac{|\mathcal{I}_{(a,b)}|}{m}$.

Chapter 3

Previous Work

Enumeration algorithms have been around for a long time. Classical results as early as 1965 [111, 88] show that a graph on n vertices contains at most $3^{\frac{n}{3}} = O(1.4423^n)$ maximal independent sets, all of which can be enumerated with polynomial delay. The area is still active. For instance, in 2007, all maximal c-isolated cliques of an m-edge graph were proved to be enumerable in $O(2.89^c \cdot c^2 \cdot m)$ time [95].

In this thesis, we are concerned with parameterized enumeration, and in particular the enumeration aspects of two parameterized problems.

3.1 Parameterized Counting

Perhaps due to the complexity of enumerating all solutions, the research has been mostly focused on counting the number of solutions. Among the first results is Arvind and Raman's proof for hardness of counting k-Cliques [11]. Later, Flum and Grohe proposed a hierarchy of parameterized counting problems [71]. They proved sample intractability results for counting k-paths and k-cycles in general directed/undirected graphs and for counting k-cliques in general graphs (all proved to be $\#W[1]$-hard), and for counting k-dominating sets (proved to be $\#W[2]$-complete), essentially showing that it is very unlikely that any of these parameterized problems has an FPT algorithm. Indeed, if all the counting problems in $\#W[1]$ are solvable in FPT time (i.e. are in $\#FPT$), then there exists an $O(2^{o(n)})$-time algorithm to evaluate the satisfiability of any n-variable 3-CNF formula, which is believed to be highly unlikely [71]. Independently, a similar framework and intractability results were also derived by McCartin [109].

Aside from the hardness results, many counting problems have been shown to be tractable through development of FPT algorithms that count the number of solutions. Such algorithms are often adaptations of the fixed-parameter tractable algorithms developed for decision versions of the problems. As a general rule, FPT algorithms based on dynamic programming are generalizable to FPT counting algorithms. In addition, a large portion of logic-based parameterized tractability results transfer to counting problems. Counting versions of graph problems definable in monadic second-order logic are fixed-parameter tractable, when parameterized by tree-width [10]. For example, this implies the fixed-parameter tractability of counting Hamiltonian cycles of a graph. With some restrictions, the tractability result extends to parameterizations using the clique-width of the input graph [46]. For the smaller class of graph problems definable in first-order logic, counting problems are fixed-parameter tractable even with respect to the local tree-width of the input graph [74]. Examples of graphs of bounded local tree-width are planar graphs and graphs of bounded valence. This result proves that, for example, counting the dominating sets of a planar graph is fixed-parameter tractable with respect to the size of the dominating sets. Furthermore, graph problems that are Fagin-definable by a first-order formula such that the relation variable occurs outside the scope of negations and existential quantifiers have fixed-parameter tractable counting versions [71]. An example of such a problem is the minimum vertex cover problem parameterized by the size of the minimum vertex cover in the input graph [71]. In addition, the counting version of the parameterized homomorphism problem is fixed-parameter tractable when parameterized by the tree-width of the input relational structures [71, 47].

Some other algorithms use the existing kernels for the decision problems; for instance, Arvind and Raman [11] demonstrated how to count the number of k-vertex covers in time $O(2^{k^2+k}k + 2^k n)$. Not every kernel can be used to count. Diaz et al. [52] introduced certain $f(k)$-sized structures, called *compactors*, which allowed for an FPT counting algorithm. The concept was demonstrated on List h-Coloring [52]. Nishimura et al. [115] showed how to obtain compactors for matching and packing problems, achieving FPT counting algorithms for these problems. The definition was later generalized by Thurley [127]. The new definition, called *counting kernels*, was described on k-Vertex Cover and k-Hitting Set examples [127].

3.2 Parameterized Enumeration

We are not aware of enumeration techniques specifically developed for parameterized problems. Nevertheless, problem-specific approaches have led to FPT enumeration algorithms for a list of problems. However, since it is usually the case that the number of solutions (or feasible solutions in case of the optimization problems) exceeds $f(k)n^{O(1)}$ for any function f, the following relaxations of the enumeration problem are considered:

Enumeration of succinct solutions. It might be the case that a restricted set of solutions can be enumerated in FPT time. In particular, it is very common to investigate enumerations of minimal/maximal solutions for *subset problems*, in which the (feasible) solutions are subsets of size at most k of some domain set of size n [65]. For example, while the number of k-vertex covers can be as large as $\binom{n}{k}$ in an n-vertex graph, it has been shown that all minimal k-vertex covers can be computed in time $O(m + k^2 2^k)$ [66]. Furthermore, FPT algorithms have been obtained to enumerate all maximal c-isolated cliques (parameterized by c) [86, 95], minimal solutions to the Minimum Quartet Inconsistency problem [80], minimal k-triangulations of a graph [90], and minimal special k-separators, called *important separators*, in both directed [40] and undirected [35] graphs.

In some cases, all minimal solutions are located in an $f(k)$-size subset of the domain. This immediately yields an FPT algorithm that enumerates all minimal solutions. The existence of such a subset, called a *full kernel* by Damaschke [48], has been proved for k-Vertex Cover and k-Bounded Hitting Set [48, 49].

Enumeration of solution representatives. Sometimes, a large number of solutions can be specified by a single representative. This is very helpful when the number of solutions cannot be bounded by $f(k)n^{O(1)}$ for any function f, but only FPT compact representations exist.

For example, Damaschke [48] obtained an $O^*(1.74^k)$ time algorithm to enumerate special descriptions for all k-vertex covers. The bound was later improved to $O^*(1.6182^k)$ by Fernau [66].

Output-sensitive enumerations. Even when the number of solutions exceeds any fixed-parameter tractable function, there can be instances with small numbers of solutions. Therefore, it is reasonable to look for enumeration algorithms whose running times depend on the number of solutions produced.

As an example of this approach, we refer to Flum and Grohe's generalization of Courcelle's theorem [70], and to its parameterization by local tree-width [74]. In both cases, the provided enumeration algorithms were proved to execute in $O(f(k)(n + z))$ time, where z denotes the size of the output; consequently, they achieve FPT enumeration algorithms when the number of solutions is bounded by $f(k)n^{O(1)}$, for some function f.

Partial enumeration of solutions. In many applications, it is not essential to enumerate all solutions; rather, finding a certain number of solutions is sufficient. In the classical setting, people have looked at $O(K \cdot n^{O(1)})$-time algorithms to produce K solutions, or to produce all the solutions with polynomial delay, thus allowing the enumeration to be stopped after a certain number of solutions is generated. With the same goal, Chen et al. [33] defined *fixed-parameter partial enumerable* problems as the set of optimization problems that have an $O(K^{O(1)} \cdot f(k)n^{O(1)})$-time algorithm to produce K "best" solutions. Our definition of partial enumeration problems, in the preliminaries chapter, is inspired by both these partial enumerations: if $\text{Partial}(R)$ is in PF for a search problem R, then there exists an $O(K^{O(1)}n^{O(1)})$-time algorithm that enumerates K solutions in $R(x)$ for any input x. When the running time is linear in terms of K, the algorithm is a polynomial-delay enumeration algorithm; note that since we have not assumed a bound on K, a running time superlinear in K does not ensure a polynomial delay between producing the $(K-1)$st and the Kth solutions. If $\text{Partial}(R)$ is in FPT for a parameterized search problem R, then there exists an $O(f(k) \cdot K^{O(1)}n^{O(1)})$-time algorithm that enumerates K solutions. Also, if $\text{Partial}(Q)$ is in FPT, when parameterized by κ, for an optimization problem Q, then there exists an $O(f(k) \cdot K^{O(1)}n^{O(1)})$-time algorithm that enumerates K best solutions for any input instance x of Q.

Chen et al. [37] raised the question of how the two classes of FPT problems and optimization problems that are fixed-parameter partial enumerable are related. In this direction, they showed how to extend two example FPT algorithms (both using dynamic programming, one based on color-coding and one based on a tree-decomposition technique) to partial enumerations. There are few FPT algorithms developed for partial versions of optimization problems. To give more insight into Chen et al.'s question, we categorize the few existing algorithms with respect to the techniques used to obtain them:

Tree decomposition. The tree decomposition technique is generally an application of dynamic programming on tree decompositions of bounded tree-width graphs [122]. As a general rule, algorithms based on dynamic programming are generalizable to partial enumerations.

Chen et al. [33] showed that the tree-decomposition technique of Alber et al. [6] to produce a k-dominating set in a planar graph can be adapted to produce K smallest k-dominating sets in an n-vertex planar graph in time $O(2^{O(\sqrt{k})}nK \log K)$.

Color-coding. The color-coding technique is another technique often combined with dynamic programming. Chen et al. [33] showed how to adapt the color-coding technique of Alon et al., which produces a k-path in a weighted graph, to compute K largest k-paths in a weighted graph in time $O(12.8^k + 6.4^k k^2 n^3 K)$. Unlike tree-decomposition algorithms, color-coding algorithms are not adaptable to FPT counting algorithms. In this specific example, counting the number of k-paths in a graph is proved to be #W[1]-complete [71],

Bounded search-trees. Chen et al. [33] developed a search-tree algorithm to enumerate K best solutions for Weighted Vertex Cover, whose goal is to find a vertex cover in an input weighted graph. Their algorithm runs in time $O(1.47^k n + 1.22^k Kn)$.

Wang and Jiang [135] developed a search tree algorithm to find K best k-weighted feedback vertex sets. The algorithm transforms the Feedback Vertex Set problem to the Feedback Edge Set problem with specific conditions. Then it enumerates K minimum-weight feedback edge sets of size k by enumerating K maximum-weight forests, in a total running time of $O(5^k kn^2 + (5^k + 3^k K)n^2 \log n)$. The best parameterized algorithm for k-Weighted Feedback Vertex Set, whose goal is to find a minimum-weight feedback vertex set of at most k vertices, runs in time $O(5^k kn^2)$ [32].

Enumeration-based algorithms. Based on an FPT enumeration of minimal vertex covers, Wang et al. [134] developed a partial enumeration algorithm that finds K best k-edge dominating sets in a weighted graph in time $O(5.6^{2k} k^4 K^2 + 4^{2k} nk^3 K)$.

The classical techniques for partial enumerations in discrete optimization problems have gone mostly unnoticed in the parameterized setting. We will describe in Chapter 6 how the classical backtracking or the classical partitioning techniques introduced by Murty and Lawler [112, 101] can be used for parameterized problems, improving the best time bounds for two of the above-mentioned problems.

3.3 Neighbour String

As Closest String is NP-complete even for binary strings [73], much of the work has focused on finding approximate solutions or on parameterized analysis of Closest String and Neighbour String.

3.3.1 Approximation Algorithms

After a line of research on approximation algorithms [15, 78, 100], Li et al. [103], Andoni et al. [9], and Ma and Sun [105] have each demonstrated a polynomial-time approximation scheme (PTAS) for Closest String, the most recent with running time $O(\ell(n\ell)^{O(\epsilon^{-2})})$.

3.3.2 Parameterized Algorithms

Parameterized complexity of Closest String has been considered with respect to the lengths of input strings, i.e. ℓ, the number of input strings, i.e. n, and the distance allocation provided, i.e. d. In most of the investigations, $|\Sigma|$ is assumed to be a constant.

There are at most $|\Sigma|^\ell$ strings of length ℓ. Therefore, when $|\Sigma|$ is a constant, Closest String is trivially in FPT with respect to ℓ. Regardless of Σ, the definition of d ensures that the input strings match in all positions except for at most nd positions [81], yielding a simple reduction rule making $\ell \leq nd$.

Gramm et al. [82] gave an integer programming formulation of the Closest String problem in $(n-1)B(n)$ variables, where the Bell number $B(n)$ is at most $n!$. Integer programs can be solved in time polynomial in the size of the problem and number of variables [102]. As a consequence, Closest String is fixed-parameter tractable (in FPT) when parameterized by n, albeit with a huge function on n. Further developments have included solutions for the special cases of $n = 3$ [81] and, for binary strings, $n = 4$ [25].

Solutions to Neighbour String have been built on a search tree algorithm of Gramm et al. [81] for Closest String (which was used to prove the fixed-parameter tractability with parameter d), and share the key ideas of careful selection of two strings s_x and s_y and branching on all acceptable substrings in the region R of positions on which s_x and s_y differ. The StringSearch algorithm of Ma and Sun [105], a modification of the algorithm of Gramm et al. [81], computes a neighbour string in time $O(n\ell + nd\binom{d+b}{b}(|\Sigma|-1)^b 4^b)$ for instances having $\min_i d_i \leq b$ and $\max_i d_i \leq d$. This algorithm generalized that of Gramm et al. [81] by considering all substrings of R in a single round rather than position by position. A series of further papers culminated in a running time of $O(n\ell + nd(\sqrt{2}|\Sigma| + \sqrt[4]{8}(\sqrt{2}+1)(1+\sqrt{|\Sigma|-1}) - 2\sqrt{2})^d)$ [136, 141, 39]; these papers made refinements in the choice of the strings s_x and s_y to aid in the analysis. Lokshtanov et al. [104] showed that there does not exist any $O(2^{o(d \log |\Sigma|)} \cdot (n\ell)^{O(1)})$ time algorithm, unless the Exponential Time Hypothesis (ETH) fails.

Recently, Chen et al. [38] added a third input string to the computations, and obtained an $O(n\ell + nd^3 6.7308^d)$-time algorithm for binary strings and an $O(n\ell + nd1.612^d(|\Sigma| +$

$\beta^2 + \beta - 2)^d)$-time algorithm for arbitrary alphabet Σ, where $\beta = \alpha^2 + 1 - 2\alpha^{-1} + \alpha^{-2}$ with $\alpha = \sqrt[3]{\sqrt{|\Sigma| - 1} + 1}$. For small $|\Sigma|$'s, this time bound is an improvement over the previous best running time, due to Chen and Wang [39]. The algorithm and analysis are both considerably more complicated than those for the approaches using two strings; as the number of strings increases, so does the number of different kinds of difference regions.

3.3.3 Counting and Enumeration

The reduction used to prove the NP-hardness of Closest String can also be used to prove that the corresponding counting problem is #P-hard [26]. The integer programming formulation of Gramm et al. can also be used to enumerate all the solutions [26], thus proving the fixed-parameter tractability of the enumeration problem with parameter n.

Obtaining a parameterized enumeration algorithm with parameter d is not possible for general input instances. Instances that are non-minimal (defined in Section 2.2.1), for which there exist neighbour strings with all the distances strictly less than the d_i's, might have as many as $\binom{\ell}{\max_i d_i}$ neighbour strings. The simplest example is an instance consisting of a single string s_1 of length ℓ and an arbitrary number as d_1. Given this constraint, we will consider an algorithm to be an enumeration algorithm if it solves Enum(P-Neighbour String) (i.e. produces all neighbour strings when the input instance is minimal).

We observe that in its original form, the $O(n\ell + nd(d+1)^d)$-time search tree algorithm of Gramm et al. [81, 82] is not an enumeration algorithm. Starting with an input string s_x, the algorithm tries to find an input string s_y such that $H(s_x, s_y) > d$. If there is no such string, then s_x is produced as the sole solution. When the set of solutions is a superset of the input strings, even if the algorithm were modified to take the union of solutions found from starting at each of the input strings, the algorithm would fail to produce any solution that is not an input. In the example of the strings $\{0010, 0100, 1001, 1111\}$ and the value $d = 3$, there are eight solutions outside the set of inputs, such as 0001 and 0011, which are not found by this algorithm.

3.4 Kemeny Rank Aggregation

The initial research on Kemeny Rank Aggregation was mostly focused on the properties of optimal aggregations and how much these properties match the expected properties in a fair voting. The ultimate goal in this research was a comparison of various versions of

RANK AGGREGATION. In the following, we briefly mention the justifications provided by Kemeny [92], Diaconis [51], and Young and Levenglick [140] for choosing KEMENY RANK AGGREGATION as the preferred RANK AGGREGATION problem.

KEMENY RANK AGGREGATION is *neutral* [51] (in the sense that its distance measure does not depend on the labels of candidates [140]), has a natural interpretation [51], and the distance between two preference lists can be efficiently computed according to its distance measure [51]. Furthermore, KEMENY RANK AGGREGATION is *consistent* [140], where a RANK AGGREGATION problem is *consistent* if for any partitioning of the input votes $\mathcal{I}$ into two multi-sets $\mathcal{I}_1, \mathcal{I}_2$ for which $\mathrm{KRA}(\mathcal{I}_1) \cap \mathrm{KRA}(\mathcal{I}_2) \neq \emptyset$, $\mathrm{KRA}(\mathcal{I})$ is precisely $\mathrm{KRA}(\mathcal{I}_1) \cap \mathrm{KRA}(\mathcal{I}_2)$. In addition, all optimal aggregations satisfy *the Condorcet Criterion*, where a total order satisfies the Condorcet Criterion if it prefers a candidate to every other candidates when the candidate is preferred by the majority to all other candidates. In fact, as mentioned in Chapter 2, optimal aggregations satisfy the Extended Condorcet Criterion, an extension of the Condorcet Criterion due to Truchon [128] as well [57].

According to Young and Levenglick, KEMENY RANK AGGREGATION is essentially the only consistent and neutral RANK AGGREGATION problem whose optimal aggregations satisfy the Condorcet Criterion [140]. Also, assuming that the votes are "noisy" versions of a single total order, where each pair of candidates had been swapped with probability $p < 0.5$, an optimal aggregation would be a most probable total order that might have produced the input votes [139].

On the negative side, KEMENY RANK AGGREGATION is computationally harder to solve compared to most of its counter-parts. In fact, based on a reduction from k-FAS [125], Bartholdi et al. [13] showed that KEMENY RANK AGGREGATION is NP-hard. Later, Dwork et al. [56, 57] proved that the problem remains NP-hard for any constant even number of votes as small as four, and for unbounded odd m's. Biedl et al. [22] discovered and fixed an error in the proof. Dwork et al. [92] used KEMENY RANK AGGREGATION in their search for an effective spam filtering method that combined the results of multiple search engines. Due to the computational intractability, they proposed to compute relaxed optimal aggregations, called locally optimal aggregations, each only constrained to have the minimum cost among the total orders at τ-distance 1 from it. Their proposed sorting-like algorithm to compute a locally optimal aggregation could start with an arbitrary total order, thus making it combinable with other spam filtering methods. Dwork et al.'s paper [56] initiated a series of algorithmic results for KEMENY RANK AGGREGATION.

3.4.1 Approximation Algorithms

The problem was shown to have an $O(n^{2.5} + mn^2)$-time 2-approximation algorithm [56]. Coppersmith et al. [44] showed that the linear-time greedy algorithm of ranking the candidates by increasing Borda count is a 5-approximation algorithm, where the Borda count for a candidates c is the sum of the number of candidates ranked higher than c over all votes. The search for approximate solutions continued with Ailon et al.'s randomized approximation algorithms of ratios $\frac{11}{7}$ and $\frac{4}{3}$ [5] based on the linear programming formulation of the problem. Later, Biedl et al. [22] showed that choosing the best vote gives a $(2 - \frac{2}{m})$-approximation ratio. Finally, Kenyon-Mathieu and Schudy developed an $O(n^{O(\frac{1}{\epsilon^4})})$-time approximation scheme for the feedback arc set problem for special weighted tournaments, which solved Kemeny Rank Aggregation as a special case [94]. Despite being a theoretical breakthrough, the resulting algorithms could not be used in practice due to large coefficients in their running times. In an attempt to develop practical approximation algorithms, Williamson and van Zuylen derived a deterministic $\frac{8}{5}$-approximation algorithm [133]. The reader is referred to a survey by Charon and Hudry [30] for a detailed list of results.

3.4.2 Parameterized Algorithms

Since approximate solutions to Kemeny Rank Aggregation can violate important properties [43], algorithms to find exact solutions have garnered significant interest. Computational experiments of Davenport and Kalagnanam [50] suggest that exact solutions can be efficiently found when input votes have specific structures. Consequently, a series of algorithms to find exact solutions, each running fast on a group of input instances, have been developed. Betzler et al. proposed the first set of these algorithms, of fixed-parameter running times of $O(2^n \cdot n^2 m)$ [20], $O(1.53^{k_t} + m^2 n)$ [20] and $O((3k_m+1)!\, k_m \log k_m \cdot mn)$ [19], where n is the number of candidates, m is the number of votes, k_t is the τ-distance of an optimal aggregation from the votes, and k_m is the maximum pairwise τ-distance of the votes. The last-mentioned running time in this set was obtained by bounding the sets of candidates that can assume each position in an optimal aggregation. Betzler et al. [20] extended this idea to the average pairwise τ-distance of votes, denoted by k_a, and the maximum difference between the positions of a particular candidate in any of the votes, denoted by r_m, yielding bounds of $O(16^{k_a} \cdot (k_a^2 \cdot m + k_a \cdot m^2 \log m \cdot n))$ and $O(32^{r_m} \cdot (r_m^2 \cdot m + r_m \cdot m^2))$ [20].

We considered $\frac{k_t}{m}$ as an average parameter tighter than k_a, and obtained an $O^*(5.823^{\frac{k_t}{m}})$-time algorithm [123], based on an algorithm for WDFAS in tournaments [120].

We also obtained algorithms of running times $O^*(1.403^{k_t})$ and $O^*(4.829^{k_m})$ [123]. Details will be included in Chapter 5. Later, a subexponential-time algorithm developed by Alon et al. [7] for WDFAS for tournaments improved the running times with respect to $\frac{k_t}{m}$, k_a, and k_m, to $O(2^{O(\sqrt{\frac{k_t}{m}}\log\frac{k_t}{m})} + n^{O(1)})$ [68]. At about the same time, Karpinski and Schudy [91] reduced P-Kemeny Rank Aggregation to WDFAS for complete digraphs with arc-weights satisfying the probability constraint (also mentioned in Chapter 2), where the weights of the arcs (a, b) and (b, a) add up to one for every pair a, b of vertices. Karpinski and Schudy [91] solved this restricted version by pushing further the idea of bounding sets of candidates (here, vertices), that can assume each position. Through an elegant analysis, they obtained an improved running time of $O(2^{O(\sqrt{\frac{k_t}{m}})} + n^{O(1)})$. Independently, a similar time bound was obtained by Feige [64] for unweighted graphs. Though most of the algorithms for parameterizations of Kemeny Rank Aggregation have benefited from its connection to WDFAS [123, 68, 91], details of the reductions differ.

Not much improvement (with respect to $\frac{k_t}{m}$) is expected, since an $O(2^{o(\sqrt{\frac{k_t}{m}})}+n^{O(1)})$-time algorithm for P-Kemeny Rank Aggregation would cause the failure of the Exponential Time Hypothesis [7]. On the other hand, since k_t has the guaranteed lower bound of $\ell = \sum_{a,b\in U} \min\{|\mathcal{I}_{(a,b)}|, |\mathcal{I}_{(b,a)}|\}$, Fernau et al. [68] studied an above-guarantee parameterization of Kemeny Rank Aggregation. The reduction to WDFAS results [36] in an $O(2^{O(k_g \log k_g)} + n^{O(1)})$-time algorithm for the above-guarantee parameter $k_g = k_t - \ell$. For an odd number of votes, the algorithm of Karpinski and Schudy [91] runs in time $O(2^{O(\sqrt{k_g})} + n^{O(1)})$. Again, an $O(2^{o(\sqrt{k_g})} + n^{O(1)})$-time algorithm for Kemeny Rank Aggregation results in the failure of the Exponential Time Hypothesis [68], thus is very unlikely to exist.

3.4.3 Kernelization

The first kernelization result for P-Kemeny Rank Aggregation is due to Betzler et al. [20]. They used two reduction rules: one fixed the orderings of candidates on which all the votes agree, and the other fixed any ordering appearing in more than k_t votes. This led to a kernel of $2k_t$ votes over at most $2k_t$ candidates [20]. Later results [21, 18] made use of the parameter k_a, which is much smaller than k_t (approximately $\frac{k_t}{m}$), obtaining partial kernels [18] (a reduction in the number of candidates but not the number of votes). The first partial kernel [21] gave a reduction to at most $9k_a + 162k_a^2$ candidates. The number of candidates was also analyzed with respect to the number n_d of pairs of candidates that are not ordered in the same way by two-thirds or more of the votes, giving a second upper bound $2n_d + 8n_d^2$ on the number of candidates. The bound in terms of k_a was later decreased to $(\frac{16}{3})k_a$ [18, 17], using a reduction rule based on the preferences of a three-fourths majority [18]: remove any candidate whose relative order with respect to every other candidate is the same in at least three-fourth of the votes. The proof immediately gives a $(\frac{16}{3})\frac{k_t}{m}$-candidate partial kernel. The reduction rule used was shown to be a special version of Condorcet-Set Rule [18], a formalization of the well-known Extended Condorcet Criterion [128].

We are not aware of any kernels for general WDFAS. There is a $k(k+2)$ vertex kernel for the weighted feedback arc set problem on tournaments [53, 7]. Also, there is a $(2+\epsilon)k$ vertex kernel for the unweighted feedback arc set problem on tournaments [16], and a $(12k^2 - 2)$ vertex kernel for the feedback arc set problem on bipartite tournaments [138], both based on detecting "transitive modules" and partitioning the input digraph based on these modules. The $(2+\epsilon)k$ vertex kernelization [16] uses a polynomial-time approximation scheme to detect modules. A simpler algorithm by Paul et al. [119] finds modules without the help of a polynomial-time approximation scheme, but gives a weaker $4k$ bound on the number of vertices. The $(2+\epsilon)k$ vertex kernel extends to constant integer weights [16]. Constant-integer weights cannot model P-Kemeny Rank Aggregation instances with parameter $\frac{k_t}{m}$: the weights are not constant, and even if the algorithm worked for non-constant integer weights, the parameter would become k_t instead of $\frac{k_t}{m}$.

3.4.4 Counting and Enumeration

To the best of our knowledge, there are only few results on counting optimal aggregations, including those obtainable by adjusting the $O^*(2^n)$-time dynamic programming of Betzler et al. [20] or the $O^*(2^{O(\sqrt{\frac{k_t}{m}})})$ subexponential-time algorithm of Karpinski and Schudy [91] to count the number of optimal aggregations. We are the first to obtain a parameterized bound (an upper bound of $4^{\frac{k_t}{m}}$) on the number of optimal aggregations, thus giving the first parameterized algorithm for Enum(P-Kemeny Rank Aggregation).

3.4.5 Variants of the Problem

Variants of the problem where the votes or optimal aggregations are not necessarily total orders have also been studied [20, 4, 27]. Defining the distance measure becomes a challenge when the preference lists are only partially specified, mainly because an incomplete preference list can be interpreted in various ways. For example, consider a TA assignment system where students should specify their preferences over eight types of TA duties:

marking, face-to-face consulting with students, consulting through electronic communication, conducting tutorials, coordinating other TAs, creating assignment solutions, creating marking schemes, and creating scripts. Two students choosing (marking, creating scripts, coordinating other TAs) and (marking, creating scripts, coordinating other TAs, conducting tutorials, face-to-face consulting) might have the same or different opinions. So, the distance between these two partially-specified preference lists depends on how we interpret them.

Furthermore, for some applications, there is a concern of defining a *proper* distance measure, where the distance between two preference lists is not zero unless the two lists are the same, or a metric distance measure. In our TA assignment example, if we decide to consider two preference lists that show no conflict, as for the (marking, creating scripts, coordinating other TAs) and (marking, creating scripts, coordinating other TAs, conducting tutorials, face-to-face consulting) preference lists, to have distance zero, our distance function will not be proper. Even if we consider such preference lists as at ϵ distance to make the function proper, we will still have a non-metric measure, since preference lists at ϵ distance of another preference list might be very far from each other, in particular since we have assumed that a preference list on a small subset of candidates is close to a broad range of other preference lists.

In order to define meaningful distance measures, people have looked at several categories of partially specified preference lists, each having a special-purpose interpretation. Examples include total orders of subsets of candidates [57, 56], orderings of candidates with possible ties [62, 4], and total orders of the most preferred k candidates [63, 4]. We do not consider these variants in this thesis.

Chapter 4

Neighbour String

In this chapter, we consider enumeration of solutions for NEIGHBOUR STRING, the problem of finding a string σ that is within given distances from n given strings. Throughout the chapter, we assume that $\mathcal{I} = \{(s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n)\}$ is the input instance, σ and s_i's are in Σ^ℓ, and d_i's are non-negative integers.

In the following sections, we describe our algorithm, CrossoverSearch, for NEIGHBOUR STRING. We also provide a modification of StringSearch [105], EnumStringSearch, to describe part of our new analysis technique[1]. Both CrossoverSearch and EnumStringSearch solve ENUM(P-NEIGHBOUR STRING) and ENUM(P-CLOSEST STRING). Consequently, as mentioned in Section 2.2.1, they both enumerate all neighbour strings in the case of minimal input distances. Since the two algorithms solve NEIGHBOUR STRING and CLOSEST STRING in general (they produce at least a single solution for non-minimal input instances), they are comparable to the existing algorithms in the literature. We are the first to present FPT algorithms for ENUM(P-NEIGHBOUR STRING) and ENUM(P-CLOSEST STRING).

Our algorithm, CrossoverSearch, makes use of new ideas in both the algorithm and its analysis. As in previous (non-enumerating) algorithms, we construct a region R on which two strings differ. In order to take advantage of all n of the input strings instead of just two or three, we take different approaches to solutions that are "close" to R and "far" from R. For the strings that are "close", we form $n+1$ representative strings such that each "close" solution is very close to one of the representatives, allowing us to handle all such solutions in $n+1$ recursive calls. The removal of the "close" strings allows us to exploit the additional structure that results on the remaining "far" strings, allowing us to improve bounds on the reduced instance. The differences between the two algorithms

[1]A short version of this chapter was published in the proceedings of IPEC 2012 [116].

will be made more clear after we describe the slightly modified version (Algorithm 3) of StringSearch [105] in Section 4.1. Further details on the ideas in CrossoverSearch along with an example are given in Section 4.2.

Our analysis can be viewed as extending the idea of considering a third string in a previous algorithm 3-String by Chen et al. [38], to the consideration of all input strings. In doing so, we avoid the detailed case analysis that was required to handle all the different subregions formed in 3-String.

Another contribution of our work is to introduce tuning constants ϵ and δ that allow us to optimize the running time of our algorithm for values of n and d that are related in different ways. This results in an overall running time of $O(n\ell + ndN_\delta(d, b, \epsilon d, n))$ for CrossoverSearch, when called for input instances having n strings of $\max_i d_i \leq d$ and $\min_i d_i \leq b$, δ, and $t_0 = \epsilon d$, where $N_\delta(d, b, t_0, n)$ is the maximum number of leaves in the produced search trees (and hence the number of solutions). Furthermore, we prove that

$$N_\delta(d, d, \epsilon d, n) \leq ((n+1)(d+1))^{\lceil \log_{(1-\frac{\delta}{2})} \epsilon \rceil} (|\Sigma| - 1)^d 5^{\lceil d(1+\epsilon+\delta) \rceil}.$$

The measure k_{min}, used in defining "close" and "far", can also be used to advantage in the analysis of EnumStringSearch. We prove a bound on the maximum number of leaves in the search tree of EnumStringSearch for input instances having n strings of $\max_i d_i \leq d$ and $d_1 \leq b$, denoted by $M(d, b, n)$. In particular, we prove that

$$M(d, d, n) \leq 2(n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil} (|\Sigma| - 1)^d 6^{\lceil d(1+\epsilon) \rceil},$$

for any $0 < \epsilon \leq 1$, and that the running time is in $O(n\ell + ndM(d, d, n))$.

Before discussing the main algorithm, CrossoverSearch, in Section 4.2, we introduce some of our analysis in a simpler example, namely our minor modification EnumStringSearch of the StringSearch algorithm of Ma and Sun [105], in Section 4.1. We will have a closer look at the running times of CrossoverSearch and EnumStringSearch compared to the previous algorithms in Section 4.3.

4.1 A New Measure for the Analysis

The algorithm EnumStringSearch showcases a simplified form of our new measure in the analysis; the algorithm itself is a minor modification of Ma and Sun's StringSearch that makes it enumerative. The algorithm proceeds by setting the "origin" string s_x to s_1. The

Algorithm 3: EnumStringSearch

```
   Require: An instance $\mathcal{I} = \langle (s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n) \rangle$
   Assume: $\forall s\ \exists 1 \leq i \leq n$ such that $H(s, s_i) \geq d_i$
1  NS ← ∅;
2  $x \leftarrow 1$;                              /* initializing $(s_x, d_x)$ to $(s_1, d_1)$ */
3  if $d_x < 0$ then return NS;
4  if $H(s_x, s_i) \leq d_i$ for all $1 \leq i \leq n$ then
5      NS ← $\{s_x\}$;                          /* $s_x$ is a neighbour string */
6  end
7  if $d_x = 0$ then return NS;             /* no more bit changes are allowed */
8  Choose $s_y \in \mathcal{I}_s$ such that $H(s_x, s_y) \geq d_y$;       /* choose a far string $s_y$ */
9  $R \leftarrow P(s_x, s_y)$;        /* the set of positions in which $s_x$ and $s_y$ differ */
10 foreach $|R| - d_y \leq k \leq d_x$ do
11     foreach $w \in \Sigma^{|R|}$ with $H(w, s_x|R) = k$ do
12         foreach $1 \leq i \leq n$ do $d'_i \leftarrow d_i - H(w, s_i|R)$;
13         $d'_x \leftarrow \min\{d'_x, d'_y\}$;       /* $s_x$ and $s_y$ are the same in the region $\overline{R}$ */
14         NS ← NS ∪ $(w \oplus_R$ EnumStringSearch$(\langle (s_1|\overline{R}, d'_1), (s_2|\overline{R}, d'_2), \ldots, (s_n|\overline{R}, d'_n) \rangle))$;
15     end
16 end
17 return NS;
```

search for neighbour strings "originates" from s_x in the sense that the algorithm searchs for bits that must be changed in s_x in order to reach a neighbour string. The algorithm thus first decides if no change is necessary, and s_x is already a solution (line 5). A region R is defined as the set of positions in which s_x differs from a second string, s_y. In Example 4.1 below, s_2 is chosen as the "far" string s_y, as it satisfies the condition (line 8) that s_x and s_y differ in at least $d_2 = 4$ positions. The algorithm tries each possible assignment of characters to the positions in R, and for each such assignment w, forms solutions by recursively solving the problem on a reduced instance consisting of the substrings formed by removing all positions in R. In lines 10 and 11, values of w are grouped by k, where $k = H(w, s_x|R)$, the distance between w and the restriction of s_x to R. The distance allotment d'_i for the reduced instance is determined by subtracting from d_i the number of positions that differ between $s_i|R$ and w (line 12). We observe that since $s_x|\overline{R} = s_y|\overline{R}$, we can set d'_x to the minimum of d'_x and d'_y (line 13).

Example 4.1. Suppose that $\mathcal{I} = \langle (000000000, 4), (111110000, 4), (100111101, 4), (011110010, 4) \rangle$ and $s_x = s_1$. Also, suppose that s_2 is chosen as the "far" string s_y at line 8. Then, R

will be the set $\{1,2,3,4,5\}$ of positions in which s_1 and s_2 differ. Consider the iteration of the loop in which w is set to 00011 (line 11). In this iteration, the d_i''s are determined as $d_2' = 4-3 = 1$, $d_3' = 4-1 = 3$, $d_4' = 4-2 = 2$, and $d_x' = d_1' = \min\{4-2, d_2'\} = 1$, meaning that if the neighbour string we are looking for equals $w = 00011$ when restricted to the region R, then its restriction to $\overline{R}$ has to be within distances $d_1' = 1$, $d_2' = 1$, $d_3' = 3$, and $d_4' = 2$ to $s_1|\overline{R}$, $s_2|\overline{R}$, $s_3|\overline{R}$, and $s_4|\overline{R}$, respectively.

The algorithm shown in Algorithm 3 differs from StringSearch only in lines 8 (where the original $>$ is replaced by $\geq$) and 14. The importance of the change to line 8 is that the search will continue even when a neighbour string is found. If the d_i's are not minimal, the algorithm will find at least one solution (if any exist), but is not guaranteed to find all solutions. To be precise, the algorithm will terminate early at line 8 if s_x is a neighbour string that is closer than d_i for each s_i.

Lemma 4.1 below shows that, despite the change to line 8, the bounds obtained [105] for StringSearch still hold. Item 3 demonstrates an upper bound on the maximum number of leaves in the search tree of EnumStringSearch for input instances having n strings of $\max_i d_i \leq d$ and $d_1 \leq b$, denoted by $M(d,b,n)$. Items 1 and 2 are intermediate lemmas and are included for future reference in our new analysis.

The proof follows from the analogue in the analysis of StringSearch by Ma and Sun [105] and the fact that $H(s_x, s_y)$ can equal d_y has no impact on those proofs. We include the proofs for completeness. A more high-level description can be found in Section 2.2.2. The first item generalizes the fact that if $|R| > d_x + d_y$, no neighbour string exists for $\mathcal{I}$; the closer $|R|$ is to $d_x + d_y$, the smaller the value of d_x', and thus the easier the reduced instance. The second item shows that at each recursive call, the size of d_x is at most half that at the previous level; we will see how to generalize this result in CrossoverSearch. The bound given by the third item will prove useful in improving the results of each algorithm, as in Theorem 4.1, where this result is used as a way to take an early exit from a recursive analysis.

Lemma 4.1.
1. $d_x' \leq \frac{d_x + d_y - |R|}{2}$ *(or, equivalently,* $|R| \leq d_x + d_y - 2d_x'$*)*
2. $d_x' \leq \frac{d_x}{2}$
3. *For all* $0 \leq b \leq d$, $M(d,b,n) \leq \binom{d+b}{b}(|\Sigma|-1)^b 4^b$.

Proof. The d_x' and d_y' computed at line 12 are equal to $d_x - H(w, s_x|R)$ and $d_y - H(w, s_y|R)$, respectively. Since d_x' is updated to $\min\{d_x', d_y'\}$, $d_x' \leq \min\{d_x - H(w, s_x|R), d_y - H(w, s_y|R)\} \leq \frac{d_x + d_y - (H(w,s_x|R) + H(w,s_y|R))}{2}$.

Since R is the region in which s_x and s_y disagree, $H(w, s_x|R) + H(w, s_y|R) \geq |R|$, and therefore, $d'_x \leq \frac{d_x+d_y-|R|}{2}$.

Since s_y is chosen such that $H(s_x, s_y) \geq d_y$, $|R| \geq d_y$. Therefore, the upper bound $\frac{d_x+d_y-|R|}{2}$ on d'_x is less than or equal to $\frac{d_x}{2}$.

Ma and Sun [105] proved an upper bound

$$\binom{d+b}{b}(|\Sigma|-1)^b 4^b$$

on the maximum number of leaves in the search tree of StringSearch for input instances having n strings of $\max_i d_i \leq d$ and $d_1 \leq b$, based on a recursive bound. As a result, in order to prove 3, it suffices to show that for all $0 \leq b \leq d$,

$$M(d,b,n) \leq \begin{cases} 1 & b = 0 \\ \sum_{0 \leq k \leq b} \binom{d+k}{k} (|\Sigma|-1)^k M\left(d, \min\left\{b-k, \frac{b}{2}\right\}, n\right) & b > 0 \end{cases} .$$

We consider an input $\mathcal{I} = \langle (s_1, d_1), \ldots, (s_n, d_n) \rangle$, with $\max_i d_i \leq d$ and $d_1 \leq b$, that maximizes the number of leaves in the search tree. To make the names consistent with those in CrossoverSearch, $d_x = d_1$ (line 2), and thus, $d_x \leq b$. The lemma holds for $b \leq 0$, as the algorithm stops at line 3 or 7 producing only one node in the search tree.

For $b > 0$, the algorithm will branch on every w with $|R| - d_y \leq H(w, s_x|R) \leq d_x$ produced at line 11. For a specific k, the number of w's with $H(w, s_x|R) = k$ is at most $\binom{|R|}{k}(|\Sigma|-1)^k \leq \binom{d_y+k}{k}(|\Sigma|-1)^k \leq \binom{d+k}{k}(|\Sigma|-1)^k$.

We claim that the number of search tree leaves in the branch for any such w is at most $M(d, \min\{b-k, \frac{b}{2}\}, n)$. An upper bound on the total number of leaves in the search tree is then found by summing over all values of k the product of $M(d, \min\{b-k, \frac{b}{2}\}, n)$ and the number of w's with $H(w, s_x|R) = k$, or

$$\sum_{0 \leq k \leq b} \binom{d+k}{k} (|\Sigma|-1)^k M\big(d, \min\big\{b-k, \frac{b}{2}\big\}, n\big)$$

as needed to complete the proof.

All that remains is to prove the claim. For any w with $H(w, s_x|R) = k$, $d'_x \leq d_x - k$. Moreover, due to Lemma 4.1, $d'_x \leq \frac{d_x}{2}$. Consequently, $d'_x \leq \min\{d_x - k, \frac{d_x}{2}\} \leq \min\{b-k, \frac{b}{2}\}$. By the definition of M, the maximum number of leaves produced by the recursive call at line 14 is $M(\max_i d'_i, d'_x, n)$ which is less than or equal to $M(d, \min\{b-k, \frac{b}{2}\}, n)$ because M is non-decreasing. □

Using a new analysis, we derive a new recursive bound for $M(d,b,n)$ in Lemma 4.2. Critical to our analysis is the value $k_{min} = \min_i H(w, s_i|R)$ (it does not appear explicitly in the algorithm), defined for a particular value of w. Unlike in the original analysis of StringSearch, where a bound was based on $k = H(w, s_x|R)$, we instead categorize different branches depending on k_{min}. Consider Example 4.1 to get a sense of the difference between the two analyses. In particular, consider the branch in which 00011 is assigned to w, which leads to $k_{min} = H(w, s_3|R) = 1$ and $k = H(w, s_x|R) = 2$. In the original analysis, based on k, the number of nodes in the search tree of the recursive call, at line 14, is estimated as $M(d, d'_x, n) = M(4,1,4)$, considering $d'_x \leq d_x - k$ in the analysis. In our analysis, based on k_{min}, the number of nodes in the tree is estimated as $M(d - k_{min}, d'_x, n) = M(3,1,4)$, using the additional fact that $\max_i d'_i \leq d - k_{min}$. The new analysis provides a tighter bound for the number of nodes for each assignment to w. Nevertheless, it requires an estimation of the number of w's at a certain minimum distance from s_i's, the true number of which depends on the actual strings.

We will use a combination of the recursive bounds of Lemmas 4.2 and 4.1, item 3, to prove the complexity of EnumStringSearch in Theorem 4.1 and Corollary 4.1. We will defer the proof of Theorem 4.1 to the end of this section, after a high-level idea of the proof is provided. Notice that the upper bound holds for every $0 < \epsilon \leq 1$:

Theorem 4.1. *Given an instance $\mathcal{I} = \langle (s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n) \rangle$ of* Neighbour String *for which there does not exist s with $H(s, s_i) < d_i$ for all $1 \leq i \leq n$, EnumStringSearch($\mathcal{I}$) returns* $\mathrm{NS}(\mathcal{I})$ *in $O(n\ell + n \max_i d_i \cdot M(\max_i d_i, d_1, n))$ time, where*

$$M(d,b,n) \leq (n(b+1))^{\lceil \lg \frac{b}{\epsilon d} \rceil} (|\Sigma| - 1)^b 2^b g\big(\lceil d(1+\epsilon) \rceil, \min\big\{\big\lceil \frac{2d(1+\epsilon)}{3} \big\rceil, b\big\}\big),$$

for all $0 \leq b \leq d$, $0 < \epsilon \leq 1$, and for $g(x,y) = \binom{x}{y} 2^y$. Furthermore, $|\mathrm{NS}(\mathcal{I})| \leq M(\max_i d_i, d_1, n)$.

Stirling's inequality will simplify the bound for Closest String, where all d_i's are equal to d, and thus $b = d$.

Corollary 4.1. *Given an instance $\mathcal{I} = \langle (s_1, d), (s_2, d), \ldots, (s_n, d) \rangle$ for which there does not exist s with $H(s, s_i) < d$ for all $1 \leq i \leq n$, EnumStringSearch($\mathcal{I}$) returns* $\mathrm{NS}(\mathcal{I})$ *in $O(n\ell + nd \cdot M(d,d,n))$ time, where*

$$M(d,d,n) \leq 2(n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil} (|\Sigma| - 1)^d 6^{\lceil d(1+\epsilon) \rceil},$$

for all $0 < \epsilon \leq 1$. Furthermore, $|\mathrm{NS}(\mathcal{I})| \leq M(d,d,n)$.

Proof. By Theorem 4.1, $M(d,d,n) \le (n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil}(|\Sigma|-1)^d 2^d \binom{\lceil d(1+\epsilon)\rceil}{\lceil \frac{2d(1+\epsilon)}{3}\rceil} 2^{\lceil \frac{2d(1+\epsilon)}{3}\rceil}$. Stirling's inequality[2] for $\binom{\lceil d(1+\epsilon)\rceil}{\lceil \frac{2d(1+\epsilon)}{3}\rceil} = \binom{\lceil d(1+\epsilon)\rceil}{\lfloor \frac{d(1+\epsilon)}{3}\rfloor}$ proves the corollary:

$$\begin{aligned}
M(d,d,n) &\le (n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil}(|\Sigma|-1)^d 2^d \left(\frac{1}{\left(\frac{1}{3}\right)^{\left(\frac{1}{3}\right)}\left(\frac{2}{3}\right)^{\left(\frac{2}{3}\right)}}\right)^{\lceil d(1+\epsilon)\rceil} 2^{\lceil \frac{2d(1+\epsilon)}{3}\rceil} \\
&\le (n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil}(|\Sigma|-1)^d 2^{d+1} \left(\frac{2^{\left(\frac{2}{3}\right)}}{\left(\frac{1}{3}\right)^{\left(\frac{1}{3}\right)}\left(\frac{2}{3}\right)^{\left(\frac{2}{3}\right)}}\right)^{\lceil d(1+\epsilon)\rceil} \\
&= (n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil}(|\Sigma|-1)^d 2^{d+1} 3^{\lceil d(1+\epsilon)\rceil} \le 2(n(d+1))^{\lceil \lg \frac{1}{\epsilon} \rceil}(|\Sigma|-1)^d 6^{\lceil d(1+\epsilon)\rceil}.
\end{aligned}$$

□

Notice that the $O^*(6^{d(1+\epsilon)})$ time bound already improves (asymptotically) the dependence on d in the previous best time bound of $O^*(6.73^d)$ [38].

At the heart of the proof of Theorem 4.1 is the recursive bound for $M(d,b,n)$ given in Lemma 4.2. The formula is generated by summing the number of search tree leaves in the branch for w, over every string w produced at line 11. For any w, we will use $v(w)$ to denote the value of variable v in the body of the loop processing that w. In particular, $k_{min}(w) = \min_i H(w, s_i|R)$.

The n factor, not present in previous bounds [81, 105], results from the fact that although the number of w's at distance c from s_x is $\binom{|R|}{c}(|\Sigma|-1)^c$, there can be as many as $n\binom{|R|}{c}(|\Sigma|-1)^c$ w's at minimum distance c from the s_i's.

Lemma 4.2. *For all* $0 \le b \le d$,

$$M(d,b,n) \le n(b+1) \cdot \max_{0\le i\le \frac{b}{2}, 0\le j\le b-i} \binom{d+b-2i}{j}(|\Sigma|-1)^j M(d-j,i,n).$$

Proof. We consider an input $\mathcal{I} = \langle (s_1,d_1),\ldots,(s_n,d_n)\rangle$, with $\max_i d_i \le d$ and $d_1 \le b$, maximizing the number of leaves in the search tree. The lemma holds for $b \le 0$, as the algorithm stops at line 3 or 7 producing only one node in the search tree.

[2] For any $0 < \alpha < \frac{1}{2}$, $\binom{x}{\lfloor \alpha x\rfloor} \le \left(\frac{1}{\alpha^{\alpha}(1-\alpha)^{(1-\alpha)}}\right)$: Stirling's upper bound for $x!$ and lower bounds for $\lfloor \alpha x\rfloor$ and $\lceil (1-\alpha)x\rceil$ proves the inequality.

For $b > 0$, the algorithm will branch on every w produced at line 11. For a specific j, we consider all w's such that $k_{min}(w) = j$. We can view each such w as being formed by choosing an input string s_i, choosing j positions in R, and choosing symbols for those positions that differ from the corresponding symbols in s_i; in total the number of such w's is thus at most $n\binom{|R|}{j}(|\Sigma| - 1)^j$. We claim that the number of search tree leaves in the branch for any such w is at most

$$A(j) = \max_{0 \le i \le \min\{b-j, \frac{d_y+b-|R|}{2}\}} M(d-j, i, n). \tag{4.1}$$

An upper bound on the total number of nodes in the search tree is then found by summing over all values of j the product of $A(j)$ and the number of w's with $k_{min}(w) = j$, or

$$\sum_{0 \le j \le b} n\binom{|R|}{j}(|\Sigma| - 1)^j A(j),$$

which can be shown by straightforward mathematical manipulations to be at most

$$n(b+1) \cdot \max_{0 \le i \le \frac{b}{2}, 0 \le j \le b-i} \binom{d+b-2i}{j}(|\Sigma| - 1)^j M(d-j, i, n), \tag{4.2}$$

as needed to complete the proof. We also used the fact that the range of i in (4.2) is a superset of its range in (4.1) as $\frac{d_y+b-|R|}{2} \le \frac{b}{2}$.

All that remains is to prove the claim. We consider an arbitrary w produced at line 11. Assuming that $j = k_{min}(w)$, we show that the number of leaves produced in the loop is no more than $A(j)$.

In the case in which $d'_x(w)$ is non-negative, the number of leaves produced at the function call at line 14 is at most $M(\max_i d'_i(w), d'_x(w), n)$. Since $\max_i d'_i(w) \le \max_i\{d_i - H(w, s_i|R)\} \le \max_i d_i - \min_i H(w, s_i|R)$, we have $\max_i d'_i(w) \le d - j$. There are thus at most $M(d-j, d'_x(w), n)$ leaves produced in the loop. We now wish to show that $M(d-j, d'_x(w), n)$ is bounded above by $A(j)$. We complete the proof by showing that $d'_x(w)$ is in the range $[0, \min\{b-j, \frac{d_y+b-|R|}{2}\}]$. By definition, $d'_x(w) \le d_x - H(w, s_x|R) \le b - k_{min}(w) = b - j$, covering the first case in the minimum. By Lemma 4.1, item 1, $d'_x(w) \le \frac{d_y+d_x-|R|}{2} \le \frac{d+b-|R|}{2}$.

For a negative $d'_x(w)$, the recursive call terminates at the second line, yielding a single node. Here we show that the number of leaves produced in the loop, i.e. 1, is bounded above by $A(j)$ by demonstrating that $1 \le M(d-j, i, n)$ for some i in the range $[0, \min\{b-j, \frac{d+b-|R|}{2}\}]$. In fact, $M(d-j, i, n) = 1$ for $i = 0$, as needed to complete the proof. □

The challenge in proving Theorem 4.1 by induction stems from the fact that the relative values of i and $\lceil\frac{2(d-j)(1+\epsilon)}{3}\rceil$ are not known for the i and j maximizing the Lemma 4.2 recursive bound. Setting u to 2 in Lemma 4.3 below allows us to correlate the relative values of b and $\lceil\frac{2d(1+\epsilon)}{3}\rceil$ with the relative values of i and $\lceil\frac{2(d-j)(1+\epsilon)}{3}\rceil$.

Lemma 4.3. *For any integer u and for all $i \in [0, \frac{b}{2}]$ and $j \in [0, b-i]$, and $d \geq b$, if $i \geq \lceil\frac{u(d-j)}{u+1}\rceil$, then $b \geq \frac{2ud}{2u+1}$ and $b \geq \lceil\frac{ud}{u+1}\rceil$.*

Proof. Since $i \geq \lceil\frac{u(d-j)}{u+1}\rceil$, the relaxed inequality $i \geq \frac{u(d-j)}{u+1}$ is also true, proving that $u(i+j)+i \geq ud$. Applying the conditions $i \leq \frac{b}{2}$ and $j \leq b-i$ to this inequality results in $ub + \frac{b}{2} \geq ud$. Therefore,

$$b \ \geq \frac{2ud}{2u+1} > \frac{2ud}{2u+2}.$$

The integrality of b ensures that $b \geq \lceil\frac{ud}{u+1}\rceil$, completing the proof. □

Another challenge is that every time the recursive bound of Lemma 4.2 is applied, an $n(b+1)$ factor is generated. However, $\log b$ applications of the bound are needed before a constant b is reached. Further refinement is possible; the recursive bound of Lemma 4.1, item 3, can be combined with the recursive bound of Lemma 4.2 to reduce the exponent of $n(b+1)$ to a constant.

We first prove a preliminary version of Theorem 4.1 that is based merely on the recursive bound of Lemma 4.2:

Theorem 4.2. *For all $0 \leq b \leq d$,*

$$M(d,b,n) \leq (n(b+1))^{\lg(b)}(|\Sigma|-1)^b 2^b g(d, \min\{\lceil\frac{2d}{3}\rceil, b\}),$$

where $g(x,y) = \binom{x}{y}2^y$.

Proof. We first show that EnumStringSearch returns NS($\mathcal{I}$), i.e. NS $=$ NS($\mathcal{I}$). To make the names consistent with those in CrossoverSearch, $d_x = d_1$ (line 2). The proof is by strong induction on d_x, with the base case $d_x \leq 0$. The statement holds for $d_x < 0$, since no string can have a negative distance from another string, and NS($\mathcal{I}$) must be $\emptyset$, as returned at line 3. The statement is also true for $d_x = 0$, since NS($\mathcal{I}$) $= \{s_x\}$ if s_x is a neighbour string, and NS($\mathcal{I}$) $= \emptyset$ otherwise, as returned at line 7.

Assuming the theorem holds for values of d_x smaller than ℓ, $\ell > 0$, it is easy to show that NS $\subseteq$ NS($\mathcal{I}$) for $d_x = \ell$. Any string $\sigma \in$ NS is inserted in NS at line 14, and thus is made up of two parts w and σ' such that $\sigma = w \oplus_R \sigma'$. Due to Lemma 4.1, item 2, the d'_x computed at line 13 is at most $\frac{d_x}{2} < \ell$. Therefore, by the induction hypothesis, $\sigma' \in \mathrm{NS}(\langle(s_1|\overline{R}, d'_1), \ldots, (s_n|\overline{R}, d'_n)\rangle)$ and thus $\sigma \in \mathrm{NS}(\mathcal{I})$, as for all i $H(\sigma', s_i|\overline{R}) \leq d'_i = d_i - H(w, s_i|R)$ implies that $H(w \oplus_R \sigma', s_i) \leq d_i$.

To prove that $\mathrm{NS}(\mathcal{I}) \subseteq \mathrm{NS}$ for $d_x = \ell$, we show that every string $\sigma \in \mathrm{NS}(\mathcal{I})$ will be added to NS at the concatenation operation at line 14 during the round w is set to $\sigma|R$ at line 11. First, we note that each $w = \sigma|R$ is generated at line 11, since $H(\sigma|R, s_x|R) \leq d_x$ and the definition of R guarantees $|R| \leq H(\sigma|R, s_x|R) + H(\sigma|R, s_y|R) \leq H(\sigma|R, s_x|R) + d_y$.

Second, $\sigma|\overline{R}$ is in $\mathrm{NS}(\langle(s_1|\overline{R}, d'_1), \ldots, (s_n|\overline{R}, d'_n)\rangle)$, since otherwise, there exists $s_i \in \mathcal{I}_s$ that makes $H(\sigma|\overline{R}, s_i|\overline{R})$ greater than $d'_i = d_i - H(\sigma|R, s_i|R)$, and the resulting $H(\sigma, s_i) > d_i$ contradicts $\sigma \in \mathrm{NS}(\mathcal{I})$. Again, since d'_x computed at line 13 is at most $\frac{d_x}{2} < \ell$, the induction hypothesis can be used, and thus, $\sigma|\overline{R}$ will be among the strings returned at the recursive call.

To see that the running time bound is met, we associate each node in the search tree with the cost of steps 10-14 performed (if any) just before the node was created plus the cost of steps 1-9 performed at the time the node is being executed. Assuming that $H(s_i, s_x)$ and $P(s_i, s_x)$ are known, the cost of a node will be in $O(n)$. The distances would normally take $O(n\ell)$ time to compute, but the algorithm can reduce the time required to $O(n \max_i d_i)$ time if it initially computes and stores the distances of every input string s_i from s_x, and updates the distances, and sets of positions in $P(s_i, s_x)$, in each round based merely on the positions decided in the round, i.e. the positions in R. Consequently, each node will add $O(n \max_i d_i)$ time to the running time, and the algorithm will consume an overall $O(n\ell + n \max_i d_i \cdot N)$ time, where N is the number of nodes in the tree. The proof is completed by the observation that all internal nodes of height two or more have at least two children, generated for $k = d_x$; therefore, N is at most four times the number of leaves, or $N \leq 4M(\max_i d_i, d_1, n)$.

It remains to prove the upper bound for $M(d, b, n)$ by strong induction on b, with the base case $b = 0$. Starting with the recursive formula of Lemma 4.2, we need to show that

$$n(b+1)\binom{d+b-2i}{j}(|\Sigma|-1)^j(n(i+1))^{\lg i}(|\Sigma|-1)^i 2^i g\big(d-j, \min\big\{\big\lceil\frac{2(d-j)}{3}\big\rceil, i\big\}\big)$$

is at most $(n(b+1))^{\lg b}(|\Sigma|-1)^b 2^b g\big(d, \min\big\{\big\lceil\frac{2d}{3}\big\rceil, b\big\}\big)$ for every $i \in [0, \frac{b}{2}]$ and $j \in [0, b-i]$. The $n(b+1)(n(i+1))^{\lg i}$ and $(n(b+1))^{\lg b}$ factors and the $(|\Sigma|-1)^j(|\Sigma|-1)^i$ and $(|\Sigma|-1)^b$ are easily cancelled out, since $i \leq \frac{b}{2}$ and $j \leq b - i$. It remains to show the following inequality for the i's and j's in the ranges:

$$\binom{d+b-2i}{j} 2^i g(d-j, \min\{\lceil \frac{2(d-j)}{3} \rceil, i\}) \quad \leq \quad 2^b g(d, \min\{\lceil \frac{2d}{3} \rceil, b\}). \tag{4.3}$$

Case 1: $i \leq \lceil \frac{2(d-j)}{3} \rceil$

$$A = \binom{d+b-2i}{j} 2^i g(d-j, \min\{\lceil \frac{2(d-j)}{3} \rceil, i\}) = \binom{d+b-2i}{j}\binom{d-j}{i} 4^i.$$

The selection of j positions from $d+b-2i$ positions can be seen as a selection of $j_1 \leq j$ positions from d positions and a selection of $j-j_1$ positions from $b-2i$ positions:

$$\begin{aligned} A &= \sum_{0 \leq j_1 \leq j} \binom{d}{j-j_1}\binom{b-2i}{j_1}\binom{d-j}{i} 4^i \leq \max_{0 \leq j_1 \leq j} 2^{b-2i}\binom{d}{j-j_1}\binom{d-j}{i} 4^i \\ &= \max_{0 \leq j_1 \leq j} 2^b \binom{d}{j-j_1}\binom{d-j}{i}. \end{aligned}$$

If $\binom{d-j}{i}$ is increased to $\binom{d-(j-j_1)}{i}$, the two combinations can be viewed as choosing $j-j_1$ from d positions, and then choosing i positions from the remainder of positions:

$$2^b \binom{d}{j-j_1}\binom{d-(j-j_1)}{i} = 2^b \binom{d}{j-j_1+i}\binom{j-j_1+i}{i} \leq 2^b g(d, j-j_1+i)$$

We notice that $g(x,y)$ increases with the increase of y until $y = \lceil \frac{2x}{3} \rceil$ and decreases afterwards. Considering that $j - j_1 + i \leq j + i \leq b$, A is proved to be less than or equal to $2^b g(d, \min\{\lceil \frac{2d}{3} \rceil, b\})$, completing the proof for the first case.

Case 2: $i > \lceil \frac{2(d-j)}{3} \rceil$

In the proofs of Case 2A and 2B below, we will use the fact that $b \geq \lceil \frac{2d}{3} \rceil$ (Lemma 4.3).

$$B = \binom{d+b-2i}{j} 2^i g(d-j, \min\{\lceil \frac{2(d-j)}{3} \rceil, i\}) = 2^i \binom{d+b-2i}{j}\binom{d-j}{\lceil \frac{2(d-j)}{3} \rceil} 2^{\lceil \frac{2(d-j)}{3} \rceil}.$$

By Stirling's inequality,

$$B \leq 2^i \binom{d+b-2i}{j} \left(\frac{1}{\left(\frac{1}{3}\right)^{\frac{1}{3}} \left(\frac{2}{3}\right)^{\frac{2}{3}}} \right)^{(d-j)} 2^{\lceil \frac{2(d-j)}{3} \rceil} \leq 2^{i+1} \binom{d+b-2i}{j} 3^{(d-j)}.$$

The function $t(j) = \binom{d+b-2i}{j}3^{-j}$ is non-decreasing for $j \leq \frac{d+b-2i+1}{4}$, and decreasing afterwards, since $\frac{t(j)}{t(j-1)} = \frac{d+b-2i-j+1}{3j} \geq 1$ if and only if $j \leq \frac{d+b-2i+1}{4}$.

On the other hand, the condition on i's, and the integrality of j, forces j to be at least $\lceil d - \frac{3}{2}(i-1)\rceil$, since $i > \lceil\frac{2(d-j)}{3}\rceil$ if and only if $j \geq \frac{2d-3i+3}{2}$.

This lower bound for j is always greater than $\frac{d+b-2i+1}{4}$, since $d - \frac{3}{2}(i-1) > \frac{d+b-2i+1}{4}$ for $i \leq \frac{b}{2}$. Therefore, the maximum of B occurs at $j = x(i) = \lceil d - \frac{3}{2}(i-1)\rceil$:

$$B \leq \binom{d+b-2i}{x(i)} 2^{i+1}3^{d-x(i)} = B(i).$$

We show that $B(i)$ is maximized at the maximum i, i.e., $i = \frac{b}{2}$. To do this, we examine the ratio $\frac{B(i+1)}{B(i)}$ and prove it to be greater than one. We will encounter functions of the form $\frac{a-\alpha i}{b-\beta i}$, in the proofs, which are decreasing, non-changing, or increasing for i depending on whether $a\beta < b\alpha$, $a\beta = b\alpha$, or $a\beta > b\alpha$.

Since $x(i+1) = \lceil d - \frac{3}{2}(i-1) - \frac{3}{2}\rceil$ and $x(i) = \lceil d - \frac{3}{2}(i-1)\rceil$, the following two cases are exhaustive:

Case 2A: $x(i+1) = x(i) - 1$

$$\frac{B(i+1)}{B(i)} = 6 \cdot \frac{\binom{d+b-2i-2}{x(i)-1}}{\binom{d+b-2i}{x(i)}} > \frac{6x(i)}{d+b-2i} \cdot \frac{d+b-2i-x(i)}{d+b-2i} \geq \frac{6x(\frac{b}{2})}{d} \cdot \frac{d-x(\frac{b}{2})}{d}$$

The last inequality comes from the fact that both $\frac{6x(i)}{d+b-2i}$ and $\frac{d+b-2i-x(i)}{d+b-2i}$ are functions of the form $\frac{a-\alpha i}{b-\beta i}$.

Since $x(\frac{b}{2})$ and $d - x(\frac{b}{2})$ add up to d, the minimum of their product occurs at one of the boundaries of $x(\frac{b}{2})$. As $x(\frac{b}{2})$ is linear in b, the boundaries of $x(\frac{b}{2})$ occur at the boundaries of b, i.e. at $b = \lceil\frac{2d}{3}\rceil$ or at $b = d$. Both produce values greater than 1, by which $\frac{B(i+1)}{B(i)}$ is proved to be greater than one.

Case 2B: $x(i+1) = x(i) - 2$

$$\frac{B(i+1)}{B(i)} = 18 \cdot \frac{\binom{d+b-2i-2}{x(i)-2}}{\binom{d+b-2i}{x(i)}} \geq 18 \cdot \left(\frac{x(\frac{b}{2})}{d}\right)^2 > 1.$$

Therefore, $i = \frac{b}{2}$ maximizes $B(i)$.

$$\begin{aligned} B &\le B\left(\frac{b}{2}\right) = \frac{g(d, d - x(\frac{b}{2}))}{2^{d-x(\frac{b}{2})}} \cdot 2^{\frac{b}{2}+1} \cdot 3^{d-x(\frac{b}{2})} \\ &\le \frac{g(d, \lceil \frac{2}{3}d \rceil)}{2^{d-x(\frac{b}{2})}} \cdot 2^{\frac{b}{2}+1} \cdot 3^{d-x(\frac{b}{2})} \\ &\le 2^b g(d, \lceil \frac{2}{3}d \rceil) 2^b g\big(d, \lceil \frac{2}{3}d \rceil\big) \end{aligned}$$

□

To reduce the exponent of $n(b+1)$ to a constant, we stop using the recursive function of Lemma 4.2 after b becomes smaller than ϵd, for a tuning constant ϵ, and then use the bound $M(d,b,n) \le \binom{d+b}{b}(|\Sigma|-1)^b 4^b$ of Ma and Sun (Lemma 4.1, item 3). As ϵ increases, the recursive depth (and hence the exponent on $n(d+1)$) decreases, as the ending condition is met sooner. Since the optimal choice will depend on the relative values of n and b, the ϵ in Theorem 4.1 can be set as best for each circumstance.

Proof of Theorem 4.1.

For b's smaller than ϵd, the result follows from Lemma 4.1, item 3:

$$\begin{aligned} M(d,b,n) &\le \binom{d(1+\epsilon)}{b}(|\Sigma|-1)^b 4^b = (|\Sigma|-1)^b 2^b g(d(1+\epsilon), b) \\ &\le (|\Sigma|-1)^b 2^b g(d(1+\epsilon), \min\{\lceil \frac{2d(1+\epsilon)}{3} \rceil, b\}) \end{aligned}$$

For larger b's, the proof is similar to the proof of Theorem 4.2, except that the induction hypothesis gives the slightly worse bound of $2^i g((d-j)(1+\epsilon), \min\{\lceil \frac{2(d-j)(1+\epsilon)}{3} \rceil, i\})$ for $M(d-j, i, n)$. Nevertheless, once $d + (d-j)\epsilon$ is substituted for the d parameter in Inequality (4.3), we will have

$$\binom{d+b-2i}{j} 2^i g((d-j)(1+\epsilon), \min\{\lceil \frac{2(d-j)(1+\epsilon)}{3} \rceil, i\}) \le 2^b g(d+(d-j)\epsilon, \min\{\lceil \frac{2(d+(d-j)\epsilon)}{3} \rceil, b\}).$$

It is easy to verify that

$$2^b g(d + (d-j)\epsilon, \min\{\lceil \frac{2(d+(d-j)\epsilon)}{3} \rceil, b\}) \le 2^b g(d(1+\epsilon), \min\{\lceil \frac{2d(1+\epsilon)}{3} \rceil, b\}),$$

as needed to complete the proof. □

4.2 Using Combinations of Input Strings

Algorithm 4: CrossoverSearch

```
   Require : An instance $\mathcal{I} = \langle (s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n) \rangle$, a real number $\delta$, an integer $t_0$,
             and $(s_x, d_x)$
   Assume : $\forall s \; \exists 1 \leq i \leq n$ such that $H(s, s_i) \geq d_i$
1  if $d_x \leq t_0$ then                       /* no change of algorithm for small $d_x$'s */
2      return EnumStringSearch($\langle (s_x, d_x), (s_1, d_1), (s_2, d_2), \ldots, (s_n, d_n) \rangle$);
3  end
4  Choose $s_y \in \mathcal{I}_s$ such that $H(s_x, s_y) \geq d_y$;
5  $R \leftarrow P(s_x, s_y)$;
6  $t \leftarrow \lfloor (1 - \frac{\delta}{2}) d_x \rfloor + 1$;
7  NS $\leftarrow$ CrossoverSearch($\mathcal{I}, \delta, t_0, (s_x, t-1)$);          /* find $R$-close solutions */
8  foreach $1 \leq i \leq n$ do
9      $\hat{s} = s_i|R \oplus_R s_x|\overline{R}$;
10     NS $\leftarrow$ NS $\cup$ CrossoverSearch($\mathcal{I}, \delta, t_0, (\hat{s}, t-1)$);    /* find $R$-close solutions */
11 end
12 foreach $\max\{|R| - d_y, t - \frac{d_x + d_y - |R|}{2}\} \leq k \leq d_x$ do
13     foreach $w \in \Sigma^{|R|}$ with $H(w, s_x|R) = k$ do
14         foreach $1 \leq i \leq n$ do $d'_i \leftarrow d_i - H(w, s_i|R)$;
15         $d'_x \leftarrow \min\{d'_x, d'_y\}$;
16         if $\min_i H(w, s_i|R) + d'_x < t$ then   /* no $R$-far solutions in this branch */
17             return CrossoverSearch($\langle (s_1, -1), \ldots, (s_n, -1) \rangle, \delta, t_0, (s_x, -1)$);  /* produce a
               fake leaf, for the sake of analysis */
18         end
19         else                                      /* find $R$-far solutions */
20             NS $\leftarrow$ NS $\cup$ ($w \oplus_R$ CrossoverSearch($\langle (s_1|\overline{R}, d'_1), \ldots, (s_n|\overline{R}, d'_n) \rangle, \delta, t_0, (s_x|\overline{R}, d'_x)$));
21     end
22 end
23 return NS;
```

In this section, we highlight the ideas in CrossoverSearch, shown in Algorithm 4. The "origin" string s_x is passed as a separate parameter here, since s_x is not always one of the input strings; rather, at times, the algorithm constructs s_x as a combination of two input strings (lines 9 and 10).

Following previous algorithms, we begin by finding a difference region R (lines 4–5).

The new approach introduced in this algorithm is the classification of all solutions into two types: an R-*close solution* is "close" to the restriction of some input string to R, and an $\overline{R}$-*close solution* is "close" to restrictions of all input strings to $\overline{R}$. For the appropriate definition of "close", each solution must be of one of these types, since if a solution is not R-close, then $H(\sigma|R, s_i|R)$ is large, and hence $H(\sigma|\overline{R}, s_i|\overline{R})$ cannot be very big, since $H(\sigma|R, s_i|R) + H(\sigma|\overline{R}, s_i|\overline{R})$ must be at most d_i.

Each R-close solution will be close to at least one of the $n+1$ strings $s_i|R \oplus_R s_x|\overline{R}$, $s_i \in \mathcal{I}_s \cup \{s_x\}$. The algorithm will find all such neighbour strings through the $n+1$ recursive calls at lines 7 and 10, each of which uses one of the $n+1$ combinations of input strings as s_x along with a small distance allotment $t-1$. It is the threshold t, then, that defines "close" to distinguish R-close and $\overline{R}$-close solutions. The smaller this threshold is, the more distances are considered "far", and hence the more solutions are $\overline{R}$-close. In Example 4.1, if $t = 3$, the solution $\sigma = 000110000$ is considered R-close since $H(s_3|R, \sigma|R) = 1 \leq t-1$. Consequently, σ will be found efficiently through the function call at line 10 for $\hat{s} = s_3|R \oplus_R s_x|\overline{R}$. Indeed, this single recursive call replaces the search for all the neighbour strings in $\{\sigma : H(s_3|R, \sigma|R) + H(s_x|\overline{R}, \sigma|\overline{R}) \leq t-1\}$. After line 11, the search is confined to solutions whose restrictions to R are at distance at least 3 from all the s_i's.

The analysis of $\overline{R}$-close solutions uses the measure k_{min} in a manner analogous to the analysis of EnumStringSearch in Section 4.1. The fake leaves produced at line 17 are for technical reasons only; they make the running time of the algorithm a nice function of the number of leaves, as otherwise, some of the time-consuming branches do not produce many leaves.

The roles of lines 2 and 6 are related to the use of tuning constants for the analysis. In EnumStringSearch a single tuning constant ϵ was confined to the analysis, used to determine when to stop the recursive calls; here we add a second tuning constant δ and, unlike in EnumStringSearch, introduce both constants into the algorithm itself. Here, t_0 plays the same role as ϵ. The additional constant δ plays a role in choosing the threshold t.

We can think of the analysis as occurring in two stages. In the first stage, we recursively reduce our bound on d_x (line 6); this is similar to the reduction by halving in EnumStringSearch (Lemma 4.1, item 2). Here instead of using $\frac{1}{2}$, we use $1 - \frac{\delta}{2}$; the bigger the value of δ, the smaller the value of t, which plays a role in defining the new d_x. The first stage ends when $d_x \leq t_0$ (line 2). In the second stage, i.e. within the instance of EnumStringSearch called at line 2, the bound is halved at each recursive call, and Lemma 4.1, item 3 is invoked to complete the analysis.

The inputs include not only the n strings and distance allotments and the tuning constants δ and t_0, but also a specified string and distance allotment pair (s_x, d_x), where s_x (as formed in line 9) is not required to be one of the input strings. To avoid an increase in the number of strings in each invocation of the algorithm, s_x is not merged into $\mathcal{I}$ at line 10; also, whenever a constructed string $\hat{s}$ is expected to be closer than s_x to the solutions, $\hat{s}$ will replace the current value of s_x (line 10).

4.2.1 Analysis of CrossoverSearch

We prove the complexity of CrossoverSearch in Theorem 4.3. The ideas in the proof of Theorem 4.1, such as the use of k_{min} and the use of the recursive bound of Ma and Sun (Lemma 4.1, item 3) for small values of b are also used in the proof of Theorem 4.3. In comparison, the parameter k_{min} appears in the algorithm (according to lines 14-17, a neighbour string σ will be produced in branches of line 20 only if $\min_i H(\sigma|R, s_i|R) + d'_x \geq t$ for $d'_x = \min\{d_x - H(\sigma|R, s_x|R), d_y - H(\sigma|R, s_y|R\}$; the value $\min_i H(w|R, s_i|R)$ will be k_{min} if w is the required $\sigma|R)$, and a more complicated condition is needed at line 12. Determining a condition that satisfies both the correctness and the required complexity is a challenge.

The ratio of t_0 to d has the same role as the tuning constant ϵ in EnumStringSearch. The proof is optimized for values of δ smaller than 0.75, since these are the only values of interest. Larger values of δ will produce $5^{\lceil d(1+\epsilon+0.75)\rceil}$ factors in the bound, already worse than the 16^d previous bound [105], also mentioned in Lemma 4.1, item 3.

Theorem 4.3. *Given an instance $\mathcal{I} = \langle (s_1, d_1), (s_2, d_2), \dots, (s_n, d_n) \rangle$, for which there does not exist s with $H(s, s_i) < d_i$ for all $1 \leq i \leq n$, and for any $0 < \delta \leq 0.75$ and $0 < \epsilon \leq 1$, CrossoverSearch$(\mathcal{I}, \delta, \epsilon d, (s_1, d_1))$ returns $\mathrm{NS}(\mathcal{I})$ in time $O(n\ell + n \max_i d_i \cdot N_\delta(\max_i d_i, d_1, \epsilon d, n))$, where*

$$N_\delta(d, b, \epsilon d, n) \leq ((n+1)(b+1))^{\lceil \log_{(1-\frac{\delta}{2})} \frac{\epsilon d}{b} \rceil} (|\Sigma| - 1)^b f(d', \min\{\lceil \frac{4d'}{5} \rceil, b\}),$$

for any $0 \leq b \leq d$, and for $f(x, y) = \binom{x}{y} 4^y$ and $d' = \lceil d + \epsilon d + \delta b \rceil$. Furthermore, $|\mathrm{NS}(\mathcal{I})| \leq N_\delta(\max_i d_i, d_1, \epsilon d, n)$.

Proof. We need to show that CrossoverSearch returns $O = \mathrm{NS}(\mathcal{I} \cup \langle (s_x, d_x) \rangle)$, i.e. $\mathrm{NS} = O$. The proof is by strong induction on d_x, with the base case $d_x \leq t_0$. The algorithm will proceed as EnumStringSearch, thus returning O, when $d_x \leq t_0$.

Assuming the theorem holds for values of d_x smaller than ℓ, $t_0 < \ell$, it is easy to show NS $\subseteq O$ for $d_x = \ell$. If σ is inserted in NS at line 7 or 10, since CrossoverSearch is called with $t-1 \leq d_x - 1 < \ell$ (line 6), σ has to be in O by the induction hypothesis. Otherwise, σ is inserted in NS at line 20, and therefore, is made up of w and σ' such that $\sigma = w \oplus_R \sigma'$. Due to Lemma 4.1, item 2, the d'_x computed at line 15 is at most $\frac{d_x}{2} < \ell$. Therefore, from the induction hypothesis, $\sigma' \in \text{NS}(\langle (s_1|\overline{R}, d'_1), \ldots, (s_n|\overline{R}, d'_n), (s_x|\overline{R}, d'_x)\rangle)$, and therefore, σ satisfies the required distance restrictions $H(w \oplus_R \sigma', s_i) \leq d_i$ for all i's:

$$H(\sigma', s_i|\overline{R}) \leq d'_i \leq d_i - H(w, s_i|R)$$

To prove the other direction, i.e. $O \subseteq$ NS, for $d_x = \ell$, we consider σ's in O based on their values of $\alpha(\sigma) = \min_{s \in \mathcal{I}_s \cup \{s_x\}} H(\sigma|R, s|R) + H(\sigma|\overline{R}, s_x|\overline{R})$. If $\alpha(\sigma) < t$, σ will be returned by one of the function calls at line 7 or 10. Otherwise, if $\alpha(\sigma) \geq t$, we show that σ will be added to NS at the concatenation operation at line 20 during the round w is set to $\sigma|R$ at line 13. We note that each $w = \sigma|R$ is generated at line 13, since $H(\sigma|R, s_x|R) \leq d_x$ and the definition of R guarantees $|R| \leq H(\sigma|R, s_x|R) + H(\sigma|R, s_y|R) \leq H(\sigma|R, s_x|R) + d_y$. Also, $\alpha(\sigma) \geq t$ implies that $t - H(\sigma|\overline{R}, s_x|\overline{R}) \leq H(\sigma|R, s_x|R)$. As $\sigma \in O$, $H(\sigma|\overline{R}, s_x|\overline{R}) \leq d'_x$, which is bounded above by $\frac{d_y+d_x-|R|}{2}$ (Lemma 4.1, item 1). Consequently, $t - \frac{d_x+d_y-|R|}{2} \leq t - H(\sigma|\overline{R}, s_x|\overline{R}) \leq H(\sigma|R, s_x|R)$.

Moreover, $\sigma|\overline{R}$ is in $\text{NS}(\langle (s_1|\overline{R}, d'_1), \ldots, (s_n|\overline{R}, d'_n), (s_x|\overline{R}, d_x)\rangle)$, since otherwise, either $H(\sigma|\overline{R}, s_x|\overline{R}) > d'_x$ or there exists $s_i \in \mathcal{I}_s$ that makes $H(\sigma|\overline{R}, s_i|\overline{R})$ greater than $d'_i = d_i - H(\sigma|R, s_i|R)$. The resulting $H(\sigma, s_x) > d_x$, $H(\sigma, s_y) > d_y$, or $H(\sigma, s_i) > d_i$, contradicts $\sigma \in O$. Again, since d'_x computed at line 13 is at most $\frac{d_x}{2} < \ell$, the induction hypothesis can be used, and thus, $\sigma|\overline{R}$ will be among the strings returned at the recursive call.

The running time bound is proved by strong induction on b. The $b \leq t_0$ case is easily verifiable, using Observation 4.1 and Lemma 4.1, item 3.

For the case $t_0 < b$, due to Lemma 4.4, it suffices to prove that

$$(n+1)\left(N_\delta(d, t-1, t_0, n) + b \cdot \max_{0 \leq i \leq \frac{b}{2}, t-i \leq j \leq b-i} \binom{d+b-2i}{j}(|\Sigma|-1)^j N_\delta(d-j, i, t_0, n)\right)$$

is bounded by the target formula. Note that the target formula is non-decreasing in d and b, and therefore, the induction hypothesis's bound for $\max_{\hat{d} \leq d, \hat{b} \leq b} N_\delta(\hat{d}, \hat{b}, t_0, n)$ is automatically less than or equal to the target formula for $N_\delta(d, b, t_0, n)$.

The $(n+1)(b+1)$ and $(|\Sigma|-1)$ factors are easily cancelled out, and we are left with the following inequality to prove for all $i \in [0, \frac{b}{2}]$ and $j \in [t-i, b-i]$, with $d' = \lceil d + t_0 + \delta b \rceil$

and $d'' = \lceil d + t_0 + \delta i \rceil$:

$$\binom{d+b-2i}{j} f(d''-j, \min\{\lceil \frac{4(d''-j)}{5} \rceil, i\}) \quad \leq \quad f(d', \min\{\lceil \frac{4d'}{5} \rceil, b\}). \tag{4.4}$$

Case 1: $b < \lceil \frac{2d'}{3} \rceil$

In this case, all the produced i's and j's satisfy $i < \lceil \frac{2(d'-j)}{3} \rceil$ (contrapositive of Lemma 4.3 for $u = 2$). By definition, $f(x, y) = 2^y g(x, y)$, and thus $f(x, \min\{\lceil \frac{4x}{5} \rceil, y\}) = 2^y g(x, \min\{\lceil \frac{2x}{3} \rceil, y\})$ when y is no more than two thirds of x. As a result, Inequality (4.3) in Theorem 4.1 can be used to prove the theorem. We should note that the induction hypothesis here gives the slightly worse bound

$$\begin{aligned} f(d''-j, \min\{\lceil \frac{4(d''-j)}{5} \rceil, i\}) \quad &\leq \quad f(d'-j, \min\{\lceil \frac{4(d'-j)}{5} \rceil, i\}) = f(d'-j, i) \\ &= \quad 2^i g(d'-j, i) = g(d'-j, \min\{\lceil \frac{2(d'-j)}{3} \rceil, i\}) \end{aligned}$$

for $N_\delta(d-j, i, t_0, n)$, and we need to substitute d' for the d parameter in Inequality (4.3) to get the desired result.

Case 2: $b \geq \lceil \frac{2d'}{3} \rceil$

Case 2A: $i \leq \lceil \frac{4(d''-j)}{5} \rceil$

In this case, we need to show that $f(d', \min\{\lceil \frac{4d'}{5} \rceil, b\})$ is an upper bound for

$$C = \binom{d+b-2i}{j} f(d''-j, \min\{\lceil \frac{4(d''-j)}{5} \rceil, i\}) = \binom{d+b-2i}{j} \binom{d''-j}{i} 4^i.$$

The parameter j is in the range $(1 - \frac{\delta}{2})b - i < j \leq b - i$. We can replace j and $-j$ with their upper bounds $b - i$ and $i - (1 - \frac{\delta}{2})b$, respectively:

$$C \quad \leq \quad \binom{d+b-2i}{b-i} \binom{d'' - b + i + \frac{\delta b}{2}}{i} 4^i \leq \binom{d+b-2i}{b-i} \binom{d'-b+i}{i} 4^i$$

To make C comparable to $f(d', \min\{\lceil \frac{4d'}{5} \rceil, b\})$, we increase $\binom{d+b-2i}{b-i}$ to $\binom{d'+b-2i}{b-i}$. In addition, we would like to set i to $\frac{b}{2}$. Fortunately, since $b \geq \lceil \frac{2d'}{3} \rceil$, the function $\binom{d'+b-2i}{b-i} \binom{d'-b+i}{i} 4^i$ is non-decreasing for i in the whole interval $0 \leq i \leq \frac{b}{2}$. Thus,

$$C \quad \leq \quad \binom{d'}{b - \frac{b}{2}} \binom{d' - b + \frac{b}{2}}{\frac{b}{2}} 2^b = \binom{d'}{b} \binom{b}{\frac{b}{2}} 2^b \leq f(d', b).$$

The equality is true since both $\binom{d'}{b-\frac{b}{2}}\binom{d'-b+\frac{b}{2}}{\frac{b}{2}}$ and $\binom{d'}{b}\binom{b}{\frac{b}{2}}$ are the number of all possible selections of b positions from d' positions.
The proof is complete, since $f(x,y)$ increases with the increase of y to the point $y = \lceil\frac{4x}{5}\rceil$ and decreases afterwards.

$$C \leq f(d',b) \leq f(d', \min\{b, \lceil\frac{4d'}{5}\rceil\}).$$

Case 2B: $i > \lceil\frac{4(d''-j)}{5}\rceil$

In this case, $b \geq \frac{8d'}{9}$ (Lemma 4.3), and we need to show that

$$D = \binom{d+b-2i}{j} f(d''-j, \min\{\lceil\frac{4(d''-j)}{5}\rceil, i\}) = \binom{d+b-2i}{j}\binom{d''-j}{\lceil\frac{4(d''-j)}{5}\rceil}4^{\lceil\frac{4(d''-j)}{5}\rceil}$$

is less than or equal to $f(d', \min\{\lceil\frac{4d'}{5}\rceil, b\})$, which by Lemma 4.3 is equal to $\binom{d'}{\lceil\frac{4d'}{5}\rceil}4^{\lceil\frac{4d'}{5}\rceil}$ in this case.

To make D comparable to $\binom{d'}{\lceil\frac{4d'}{5}\rceil}4^{\lceil\frac{4d'}{5}\rceil}$, we would like to set i to $\frac{b}{2}$. However, the expression in its current form is not maximized at $i = \frac{b}{2}$, and we need to find an upper bound with this property.
By Stirling's inequality,

$$\begin{aligned} D &\leq \binom{d+b-2i}{j}\left(\frac{1}{(\frac{4}{5})^{\frac{4}{5}}(\frac{1}{5})^{\frac{1}{5}}}\right)^{d''-j} \cdot 4^{\frac{4}{5}}\left(4^{\frac{4}{5}}\right)^{d''-j} = 4^{\frac{4}{5}}\binom{d+b-2i}{j}5^{d''-j} \\ &\leq 4^{\frac{4}{5}}\binom{d''+b-2i}{j}5^{d''-j}. \end{aligned}$$

The function is increasing to the point $j = \lfloor\frac{(d''+b-2i+1)}{6}\rfloor$ and decreasing afterwards. On the other hand, j has to be greater than $(1-\frac{\delta}{2})b - i$, and the values of the function we are interested in are in the decreasing section, if $\delta < \frac{3}{4}$. Thus,

$$\begin{aligned} D &\leq 4^{\frac{4}{5}}\binom{d''+b-2i}{\lfloor(1-\frac{\delta}{2})b-i\rfloor+1}5^{d''-(\lfloor(1-\frac{\delta}{2})b-i\rfloor+1)} < \binom{d''+b-2i}{b-i}5^{d''-\lfloor(1-\frac{\delta}{2})b-i\rfloor} \\ &\leq \binom{d''+b-2i}{b-i}5^{d'-b+i}. \end{aligned}$$

Therefore, it suffices to show that

$$\binom{d'+b-2i}{b-i}5^{d'-b+i} \leq \binom{d'}{\lceil\frac{4d'}{5}\rceil}4^{\lceil\frac{4d'}{5}\rceil}. \tag{4.5}$$

The function $h(i) = \binom{d'+b-2i}{b-i}5^{d'-b+i}$ is increasing for $0 \leq i \leq \frac{b}{2}$, since $b \geq \lceil\frac{2d'}{3}\rceil$. Therefore, $h(i)$ reaches its maximum value at $i = \frac{b}{2}$. Thus,

$$D \leq \binom{d'}{\frac{b}{2}}5^{d'-\frac{b}{2}} = \binom{d'}{d'-\frac{b}{2}}5^{d'-\frac{b}{2}}.$$

The function $r(i) = \binom{d'}{i}5^i$ is non-decreasing as long as $i \leq \frac{(5d'+1)}{6}$, and non-increasing for larger i's. Not every value for $d'-\frac{b}{2}$ will be produced, however. We know that $b \geq \frac{8d'}{9}$ in this case, and hence, $d' - \frac{b}{2} \leq \frac{5d'}{9}$ and

$$D \leq \binom{d'}{\lfloor\frac{5d'}{9}\rfloor}5^{\lfloor\frac{5d'}{9}\rfloor}.$$

It remains to show that $\binom{d'}{\lfloor\frac{5d'}{9}\rfloor}5^{\lfloor\frac{5d'}{9}\rfloor} \leq \binom{d'}{\lceil\frac{4d'}{5}\rceil}4^{\lceil\frac{4d'}{5}\rceil}$, which is true based on Stirling's inequalities.

□

Again, Stirling's inequality simplifies the bound for the special case of CLOSEST STRING, where $b = d$.

Corollary 4.2. *Given an instance $\mathcal{I} = \langle(s_1, d), (s_2, d), \ldots, (s_n, d)\rangle$, for which there does not exist s with $H(s, s_i) < d$ for all $1 \leq i \leq n$, and for any $0 < \delta \leq 0.75$ and $0 < \epsilon \leq 1$, CrossoverSearch($\mathcal{I}, \delta, \epsilon d, (s_1, d)$) returns $\mathrm{NS}(\mathcal{I})$ in time $O(n\ell + nd \cdot N_\delta(d, d, \epsilon d, n))$, where*

$$N_\delta(d, d, \epsilon d, n) \leq ((n+1)(d+1))^{\lceil \log_{(1-\frac{\delta}{2})}\epsilon\rceil}(|\Sigma|-1)^d 5^{\lceil d(1+\epsilon+\delta)\rceil}.$$

Furthermore, $|\mathrm{NS}(\mathcal{I})| \leq N_\delta(d, d, \epsilon d, n)$.

The bound is mainly derived from the recursive functions in Observation 4.1 and Lemma 4.4. If $b \leq t_0$, the function call at line 2 makes the algorithm run EnumStringSearch with the additional string s_x, thus producing no more than $M(d, t_0, n+1)$ leaves.

Observation 4.1. *For all* $0 \leq t_0 \leq d, 0 \leq b \leq t_0$*,* $N_\delta(d, b, t_0, n) \leq M(d, b, n+1)$*.*

The proof for larger values of b will use much of the analysis appearing in the proof of Lemma 4.2.

Lemma 4.4. *For every* $0 \leq t_0 \leq d$*,* $t = \lfloor (1 - \frac{\delta}{2}) b \rfloor$*, and all* $t_0 < b \leq d$*,* $N_\delta(d, b, t_0, n)$ *is less than or equal to*

$$\max \begin{cases} (n+1)\left(N_\delta(d, t-1, t_0, n) + b \cdot \max\limits_{0 \leq i \leq \frac{b}{2}, t-i \leq j \leq b-i} \binom{d+b-2i}{j}(|\Sigma|-1)^j N_\delta(d-j, i, t_0, n)\right) \\ \max\limits_{\hat{d} \leq d, \hat{b} \leq b} N_\delta(\hat{d}, \hat{b}, t_0, n) \end{cases}.$$

Proof. We use strong induction on b, considering the instance $\mathcal{I}$ and values (s_x, d_x) that maximize the number of leaves in the search tree. When $d_x < b$, the number of leaves is bounded by $N_\delta(\max_i d_i, d_x, t_0, n)$, covered by the second line in the recursive formula.

From now on, we can assume that $d_x = b$, and set $d' = \max_i d_i$. We let s_y be the string chosen at line 4; R will be $P(s_x, s_y)$. The algorithm will make $n+1$ function calls at lines 7 and 10, each producing at most $N_\delta(d', t-1, t_0, n)$ nodes. It will then branch on every w produced at line 13. We claim that for any produced w there exist $0 \leq j \leq b$ and $s \in \mathcal{I}_s \cup \{s_x\}$ such that $H(w, s|R) = j$ and the number of leaves reached from w is at most

$$A(j) = \max_{\max\{t-j, 0\} \leq i \leq \min\{b-j, \frac{d_y+b-|R|}{2}\}} N_\delta(d'-j, i, t_0, n).$$

In total, the number of w's mapped to a certain j and s is at most $\binom{|R|}{j}(|\Sigma|-1)^j$.

An upper bound on the total number of leaves in the search tree is then found by the counts for lines 7, 10, and 20, for a total of

$$(n+1) \cdot N_\delta(d', t-1, t_0, n) + \sum_{0 \leq j \leq b} (n+1)\binom{|R|}{j}(|\Sigma|-1)^j A(j),$$

which by straightforward mathematical manipulations is at most

$$\begin{aligned} &(n+1)\left(N_\delta(d', t-1, t_0, n) + b \cdot \max_{0 \leq i \leq \frac{b}{2}, t-i \leq j \leq b-i} \binom{d_y+b-2i}{j}(|\Sigma|-1)^j N_\delta(d'-j, i, t_0, n)\right) \\ \leq\ &(n+1)\left(N_\delta(d, t-1, t_0, n) + b \cdot \max_{0 \leq i \leq \frac{b}{2}, t-i \leq j \leq b-i} \binom{d+b-2i}{j}(|\Sigma|-1)^j N_\delta(d-j, i, t_0, n)\right). \end{aligned}$$

All that remains is to prove the claim. We consider an arbitrary w produced at line 13.

In the case in which $d'_x(w)$ is non-negative and $\min_i H(w, s_i|R) + d'_x(w) \geq t$, we demonstrate that the number of leaves produced in the loop, which is no more than $N_\delta(\max_i d'_i(w), d'_x(w), t_0, n)$ in this case, is at most $A(j)$ for $j = k_{min}(w)$. Since $\max_i d'_i(w) \leq \max_i\{d_i - H(w, s_i|R)\}$, we have $\max_i d'_i(w) \leq d' - j$. There are thus at most $N_\delta(d' - j, d'_x(w), t_0, n)$ leaves produced at the function call at line 20. We complete the proof by showing that $d'_x(w)$ is in the range $[\max\{t - j, 0\}, \min\{b - j, \frac{d_y+b-|R|}{2}\}]$. The assumptions for this case imply the lower bound $\max\{t - j, 0\} \leq d'_x(w)$. In addition, by definition, $d'_x(w) \leq d_x - H(w, s_x|R) \leq b - j$, covering the first case in the upper bound. By Lemma 4.1, item 1, $d'_x(w) \leq \frac{d_y+d_x-|R|}{2} \leq \frac{d_y+b-|R|}{2}$.

For a negative d'_x or for $\min_i H(w, s_i|R) + d'_x$ smaller than t, the algorithm will produce a single node. Here we show that the number of leaves produced in the loop, i.e. 1, is bounded above by $A(j)$ for $j = k$, i.e. $H(w, s_x|R)$. We demonstrate that $1 \leq N_\delta(d' - k, i, t_0, n)$ for some i in the range $[\max\{t - k, 0\}, \min\{b - k, \frac{d_y+b-|R|}{2}\}]$. In fact, we prove that $i = \max\{t - k, 0\}$ will make $N_\delta(d' - k, i, t_0, n) \geq 1$ by showing that there exists an instance $\mathcal{I} = \langle(s_1, d_1), \ldots, (s_n, d_n)\rangle$ and (s_x, d_x) of $\max_i d_i \leq d' - k$ and $d_x \leq i \leq d' - k$ for which the distances $d_1, \ldots, d_n$ are minimal. Consider the instance $s_x = s_1 = s_2 = \ldots s_{n-1} = 0^{2i}$, $s_n = 1^{2i}$, and $d_1 = \ldots = d_n = d_x = i$, for $i = \max\{t - k, 0\}$. Since i is non-negative and is less than or equal to $d' - k$, as $t \leq d_x \leq d'$ follows from the definition of t (line 6), the distance requirements are met for any string with i zeros and i ones. Furthermore, no string can have a distance smaller than i from both s_1 and s_n, and thus the instance is minimal. The proof is not yet complete, since $i \in [\max\{t-k, 0\}, \min\{b-k, \frac{d_y+b-|R|}{2}\}]$ only if $\max\{t - k, 0\} \leq \min\{b - k, \frac{d_y+b-|R|}{2}\}$, which is true because of the fact that $t \leq d_x = b$, by the definition of t (line 6), and the conditions on k (line 12). □

4.3 Concluding Remarks

We presented an $O^*(5^{(1+\lambda)d} \cdot (|\Sigma| - 1)^d)$-time algorithm, for any $\lambda > 0$, called CrossoverSearch, to solve Enum(P-Neighbour String). For binary strings, the time bound achieved is an asymptotic improvement over the previous best running time of $O^*(6.73^d)$ [38] for finding a single solution for Closest String instances (where all distance allocations are equal to d).

We also gave a new analysis for the StringSearch algorithm of Ma and Sun [105], and showed how it could solve Enum(P-Neighbour String) after a slight modification. Our analysis showed a time bound of $O^*(6^{d(1+\epsilon)} \cdot (|\Sigma| - 1)^d)$, for arbitrary $\epsilon > 0$. The previous time bound of $O^*(16^d \cdot (|\Sigma| - 1)^d)$ Ma and Sun [105] also holds for the modified algorithm.

The new time bounds show how much the base of the exponential part can be pushed, both in the enumerative version of StringSearch [105] and in our new algorithm. The polynomial factors in $O^*(5^{(1+\lambda)d} \cdot (|\Sigma| - 1)^d)$ and $O^*(6^{d(1+\epsilon)} \cdot (|\Sigma| - 1)^d)$ depend on λ and ϵ, and become large when the bases get closed to 5 or 6, respectively.

Aside from theoretical improvements, the new time bound of $O^*(5^{(1+\lambda)d})$ (for binary strings) is an indication of the effectiveness of our approach. In particular, according to our intermediate bounds in terms of d and $\min_i d_i$, the time bound for CrossoverSearch reaches its maximum for $\frac{\min_i d_i}{d} = \frac{2}{3}$; in comparison, the intermediate time bound for EnumStringSearch reaches its maximum at $\frac{\min_i d_i}{d} = \frac{4}{5}$. As a consequence, we expect the ideas in CrossoverSearch to be particularly useful for instances with $\frac{2}{3} < \frac{\min_i d_i}{d} < \frac{4}{5}$.

Chapter 5

Kemeny Rank Aggregation

In this chapter, we consider Kemeny Rank Aggregation, the problem of finding an *optimal aggregation* σ that minimizes the τ-distance from a given multiset $\mathcal{I}$ of m total orders, denoted by k_t. To be more precise, we study the parameterization of Kemeny Rank Aggregation by $\frac{k_t}{m}$, called P-Kemeny Rank Aggregation, and the enumeration problem associated with it.

Throughout the chapter, we use $\mathcal{I}$ to denote a multi-set of m total orders in Total(U) for an n-element domain set U, use σ to denote an optimal aggregation of $\mathcal{I}$, and use k_t to denote $\tau(\sigma, \mathcal{I})$.

In the following sections, we describe our $O^*(1.403^{k_t})$-time algorithm[1] for P-Kemeny Rank Aggregation, our $O^*(4^{\frac{k_t}{m}})$-time algorithm[2] for Enum(P-Kemeny Rank Aggregation), and partial kernelizations[3] for P-Kemeny Rank Aggregation and Enum(P-Kemeny Rank Aggregation). Our first algorithm, improving the previous best running time of $O^*(1.53^{k_t})$ by Betzler et al. [20], was later outperformed by $O^*(2^{O(\sqrt{\frac{k_t}{m}}\log\frac{k_t}{m})})$-time [7, 68] and $O^*(2^{O(\sqrt{\frac{k_t}{m}})})$-time [91] algorithms for P-Kemeny Rank Aggregation. We include the result for completeness. We do not include our algorithms of running times $O^*(4.829^{k_m})$ and $O^*(5.823^{\frac{k_t}{m}})$ [123] in this thesis, since their running times are already improved by our $O^*(4^{\frac{k_t}{m}})$-time enumeration algorithm in this chapter.

[1]A preliminary version of this algorithm was published in the proceedings of IWPEC 2009 [123].
[2]A short version of this algorithm and its analysis will appear in the proceedings of WADS 2013 [117].
[3]These partial kernelizations are submitted to MFCS 2013.

Existing algorithms for parameterizations of Kemeny Rank Aggregation mostly use a well-known reduction (mentioned in Section 2.3.1) to Weighted Feedback Arc Set on complete digraphs. The most recent parameterized algorithm due to Karpinski and Schudy [91] uses the property that the arc weights in the reduced graphs satisfy the probability constraint (i.e. the weights of the arcs (a, b) and (b, a) add up to one for every pair of vertices a, b). In the analysis of our enumeration algorithm, we make use of the additional property that the arc weights satisfy the triangle inequality [133].

Our enumeration algorithm is based on properties of pairs of vertices that can be adjacent in minimum feedback arc sets. It neglects other properties of minimum feedback arc sets to the point that it actually enumerates locally minimum feedback arc sets (corresponding to locally optimal aggregations), the weight (resp., the τ-distance) of which does not decrease by changing the order of two adjacent vertices (resp., candidates). Therefore, the resulting parameterized upper bound on the number of locally minimum feedback arc sets, though restricted to special graph classes, is quite unexpected.

We prove that there are no more than $4^{\frac{k_t}{m}}$ locally optimal aggregations, all of which can be found in $O(nm + 4^{\frac{k_t}{m}} \cdot n^{\mu})$ time. We are the first to provide parameterized upper bounds on the number of optimal aggregations. We are not aware of any upper bound on the number of locally optimal aggregations prior to this bound. Our upper bounds for the number of (locally) optimal aggregations is tight, since there are instances with exactly the same number of (locally) optimal aggregations. For example, the instance consisting of $\frac{m}{2}$ copies of the total order over $\{u_1, u_2\}$ that orders u_1 before u_2, and $\frac{m}{2}$ copies of the total order over $\{u_1, u_2\}$ that orders u_2 before u_1 has 2 (locally) optimal aggregations, which equals $4^{\frac{k_t}{m}} = 4^{\left(\frac{(\frac{m}{2})}{m}\right)}$ in this example.

Furthermore, based on the Extended Condorcet Criterion, we reduce P-Kemeny Rank Aggregation instances to independent P-Kemeny Rank Aggregation subinstances on at most $\frac{4k_t}{m}$ candidates. The reduction is a partial kernelization for P-Kemeny Rank Aggregation. Our analysis relies on the structure of a minimum feedback arc set in an associated complete digraph. There is no problem caused by our not having access to an actual minimum feedback arc set, as we are not incorporating the structure into our reduction rules. We next add a new reduction rule to handle subinstances of size at most $\frac{2}{\epsilon}$, for arbitrary $\epsilon > 0$, achieving an $(2 + \epsilon)\frac{k_t}{m}$ candidate partial kernel, improving Betzler et al.'s partial kernel over at most $\frac{16}{3}(\frac{k_t}{m})$ candidates [18]. These reduction rules also give $(2 + \epsilon)k$-vertex kernels for WDFAS instances whose arc-weights satisfy the triangle and probability constraints, where k is the weight of a minimum-weight feedback arc set in the input graph. Unlike the $(2 + \epsilon)k$ vertex kernel of Bessy et al. [16] (for constant integer arc-weights), our reduction does not rely on the complex polynomial-time approximation scheme of Kenyon-Mathieu and Schudy [94]. We use simple reduction rules. Note that Bessy et al.'s kernelization works for constant-integer weighted tournaments, and our kernelization works for special weighted complete digraphs (equivalent to special weighted tournaments); the graph instances for the two algorithms are completely disjoint.

We describe the $O^*(1.403^{k_t})$-time algorithm for P-Kemeny Rank Aggregation in Section 5.1. Before presenting the algorithm for Enum(P-Kemeny Rank Aggregation), we highlight new ideas, including the introduction of a concise representation of feedback arc sets based on their adjacent pairs, in Sections 5.2.1 to 5.2.4. In Sections 5.2.5 and 5.2.6, we present the algorithm and proofs of its correctness and analysis. A matching lower bound is presented in Section 5.2.7. The kernelization results are presented in Sections 5.3.1 to 5.3.2.

5.1 The $O^*(1.403^{k_t})$-time Algorithm

5.1.1 The Tournament Majority Graph

In this section, we introduce a notion of tournament majority graphs, based on which we give a reduction from P-Kemeny Rank Aggregation to WDFAS for tournaments. Prior to this, P-Kemeny Rank Aggregation was known to reduce to WDFAS, but the zero-weight arcs in the reduced instances (corresponding to pairs of candidates equally-preferred in both directions) had left the impression that the more efficient algorithms solving WDFAS in tournaments could not be used for P-Kemeny Rank Aggregation.

Definition 5.1. *A* tournament majority graph *of a multiset $\mathcal{I}$ of total orders in Total(U) is a weighted tournament graph whose set of vertices is U, whose set of arcs is a superset of $w_{>\frac{1}{2}}$ and a subset of $w_{\geq\frac{1}{2}}$, and the weight of an arc (a,b) is $w(a,b)-w(b,a)$, where w is the weight function $w(a,b)=\frac{|\mathcal{I}(a,b)|}{m}$.*

The following observation gives a reduction of P-Kemeny Rank Aggregation to WDFAS restricted to tournament graphs; the idea is similar to the previous reduction to WDFAS instances mentioned in Section 2.3.3 and also to a previous reduction by Dwork et al. [56]; the main difference is that here the reduced instances are tournament graphs.

Observation 5.1. *A total order σ is in* KRA$(\mathcal{I})$ *if and only if $E(\mathrm{TM})\setminus\sigma$ is a minimum-weight feedback arc set for an arbitrary tournament majority graph* TM *of $\mathcal{I}$.*

Proof. Suppose that $\mathcal{I}$ is a multiset of m total orders in Total(U). Let $w : U \times U \mapsto R$ be the function $w(a,b) = \frac{|\mathcal{I}(a,b)|}{m}$. We know that a total order $\sigma \in$ Total(U) is in KRA$(\mathcal{I})$ if and only if rev(σ) is a minimum-weight feedback arc set in the complete digraph G with the vertex set U and the arc weight function w (Observation 2.7 in Section 2.3.3).

We establish a connection between feedback arc sets in TM and in G. Let ℓ denote $\sum_{a,b\in U, a\neq b} \min\{\frac{|\mathcal{I}(a,b)|}{m}, \frac{|\mathcal{I}(b,a)|}{m}\}$. We show for an arbitrary total order $\sigma \in$ Total(U) that $E(\mathrm{TM}) \setminus \sigma$ is a feedback arc set of weight w_1 in TM if and only if rev(σ) is a feedback arc set of weight $w_2 = w_1 + \ell$ in G. Consequently, rev(σ) is a minimum-weight feedback arc set in G if and only if $E(\mathrm{TM}) \setminus \sigma$ is a minimum-weight feedback arc set in TM, as required.

Since σ is a total order, $E(\mathrm{TM}) \setminus \sigma$ is always a feedback arc set for TM: removing the arcs in $E(\mathrm{TM}) \setminus \sigma$ leaves a subset of arcs in TM that follow the orderings in σ. For the same reason, rev(σ) is always a feedback arc set for G. Therefore, we only need to worry about the weights.

Every arc in $F_{\mathrm{TM}} = E(\mathrm{TM}) \setminus \sigma$ is in $F_G = \mathrm{rev}(\sigma)$, but some of the arcs in F_G might not be in F_{TM}. To compare total weights of F_{TM} and F_G in TM and G, respectively, we check the weight of each $(a,b) \in F_G$ in both TM and G and check whether $(a,b) \in F_{\mathrm{TM}}$.

The weight of (a,b) is always $\frac{|\mathcal{I}(a,b)|}{m}$ in G. When $|\mathcal{I}(a,b)| \geq |\mathcal{I}(b,a)|$, (a,b) is also in F_{TM}, and thus it is contributing a weight of $w(a,b) - w(b,a) = \frac{|\mathcal{I}(a,b)|}{m} - \frac{|\mathcal{I}(b,a)|}{m}$ to the weight of F_{TM}. Otherwise, it is contributing a weight of zero to the weight of F_{TM}. Therefore, the weight added for such an arc (a,b) to F_{TM} is always $\min\{\frac{|\mathcal{I}(a,b)|}{m}, \frac{|\mathcal{I}(b,a)|}{m}\}$ less than the weight added to F_G.

Overall, the total weight of the arcs in F_G will be ℓ plus the total weight of the arcs in F_{TM}, as required to complete the proof. □

We show how to adapt a search tree algorithm due to Raman and Saurabh [120] to find a minimum-weight feedback arc set for a tournament majority graph. The algorithm was originally designed for tournaments with arc weights greater than or equal to one; however, we demonstrate that the algorithm can be used for general weights if the search is confined to feedback arc sets that have no more than e arcs. This variant is especially useful for finding a minimum-weight feedback arc set in tournament majority graphs, since although these graphs can have zero-weight arcs, we will show that they have a minimum-weight feedback arc set with a small number of arcs.

In fact, the only thing we change in the algorithm of Raman and Saurabh [120] is the termination condition. The original algorithm terminated once the set of fixed arcs formed a feedback arc set of weight at most k, the weight of fixed arcs exceeded k, or the set of fixed arcs was not acyclic thus already ensured not to be a minimal feedback arc set; in the new variant, the algorithm terminates once the fixed arcs form a feedback arc set of cardinality at most e, the number of fixed arcs exceeds e, or the set of fixed arcs is not acyclic. The analysis of the algorithm remains the same, except that the parameter is changed to the number of arcs rather than the weight of a minimum-weight feedback arc set. We do not repeat the proof here, but include the algorithm (Algorithm 5) since we will use a modification of it to obtain our algorithm.

Algorithm 5: MinFas

```
Require: $G, e$
1 $O \leftarrow$ BoundedSearchTree1$(G, \emptyset, e)$;
2 return $F \in \{E(G) \setminus \sigma : \sigma \in O\}$ that has the minimum weight$(F)$;
```

Algorithm 6: BoundedSearchTree1

```
Require: $G, L, e$
1  if $G$ does not have a $C_3$ then                                  /* no cycles remain */
2      return $\{E(G)\}$;
3  else if $|L| > e$ then                                        /* cannot afford more arcs */
4      return $\emptyset$;
5  else if $G$ has a $C_3$ cycle $C$, with $E(C) \cap L \neq \emptyset$ then
6      if $E(C) \subseteq L$ then return $\emptyset$;                         /* L has a cycle */
7      else
           $\mathcal{L} \leftarrow \{L \cup S : S \subseteq E(C), S$ is a minimal FAS for $C,$ and $S \cup L$ has no cycle$\}$;
8  else if $G$ has a $C_4$ cycle $C$, then
9      $\mathcal{L} \leftarrow \{L \cup S : S \subseteq E(C), S$ is a minimal FAS for $C,$ and $S \cup L$ has no cycle$\}$;
10 else                                         /* $C_3$'s in $G$ do not have common arcs */
11     let $C$ be a $C_3$ in $G$;
12     let $e$ be a minimum-weight arc in $E(C)$;
13     $\mathcal{L} \leftarrow \{L \cup \{e\}\}$;
14 return $\bigcup_{L' \in \mathcal{L}}$ BoundedSearchTree1$((V(G), (E(G) - L') \cup \mathrm{rev}(L')), L', e)$;
```

Lemma 5.1. *Suppose that G is a weighted tournament graph and e is a positive integer. Then, MinFAS(G, e) returns a minimum-weight feedback arc set of G with at most e arcs, if one exists, in time $O^*((1+\sqrt{2})^e) \approx O^*(2.415^e)$.*

MinFAS(G, e) essentially enumerates all minimal feedback arc sets of cardinality at most e and returns one that is of minimum weight. To do so, it uses the recursive algorithm

BoundedSearchTree1 which gradually decides on the arcs in the feedback arc set and uses a set L to keep track of the arcs put in the feedback arc set so far.

The algorithm uses the following property for its termination condition: a tournament graph is acyclic when it does not have a C_3; as a result, a set of arcs L is a minimal feedback arc set for a tournament graph $G = (V(G), E(G))$ if $G' = (V(G), (E(G) \setminus L) \cup \text{rev}(L))$ does not have a C_3 [120]. Each branch is therefore stopped when either no C_3 remains after reversing the arcs in L (line 1), and thus L is a minimal feedback arc set [120], or either the number of arcs in L is already larger than e (line 3) or some of the arcs in L form a cycle (line 6), and thus L cannot be completed to a minimal feedback arc set of cardinality at most e. In each node of the search tree, the algorithm branches on all minimal sets of arcs whose removal remove a C_4 (line 8). If no C_4 exists (line 10), the C_3's in the tournament graph are all disjoint [120], making it easier to branch on all minimal sets of arcs whose removal removes all C_3's (lines 11-13).

5.1.2 The Algorithm

In this section, we show how to improve the previous best running time of $O^*(1.53^{k_t})$ due to Betzler et al. [20] to $O^*(1.403^{k_t})$.

We base our analysis on the cardinality of dirty($\mathcal{I}$), the set of pairs that are not ordered the same in the total orders in $\mathcal{I}$. We use this number to bound the number of arcs in minimum-weight feedback arc sets of TM.

Lemma 5.2. *Suppose that $\mathcal{I}$ is a multiset of $m \geq 3$ total orders and* TM *is a tournament majority graph of $\mathcal{I}$. Also, suppose that e is the number of arcs in a minimum-weight feedback arc set in* TM*. Then $|\text{dirty}(\mathcal{I})| + e \leq k_t$.*

Proof. Due to Observation 5.1, a minimum-weight feedback arc set in TM is always equal to $E(\text{TM}) \setminus \sigma$ for some $\sigma \in \text{KRA}(\mathcal{I})$.

By the definition of $\mathcal{TM}$, $E(\text{TM}) \subset w_{\geq \frac{1}{2}}$. Therefore, each of the pairs $(a, b) \in E(\text{TM}) \setminus \sigma$ indicates that σ opposes the ordering of $\{a, b\}$ suggested by the majority. Also, by the definition of dirty pairs, for each of the dirty pairs, including the dirty pairs not in $E(\text{TM}) \setminus \sigma$, there exists a total order in $\mathcal{I}$ that disagrees with the pair's ordering in σ. The number of such pairs is at least $|\text{dirty}(\mathcal{I})| - |E(\text{TM}) \setminus \sigma|$. Therefore, $k_t = \tau(\sigma, \mathcal{I})$ is at least $\lceil \frac{m}{2} \rceil \cdot |E(\text{TM}) \setminus \sigma| + (|\text{dirty}(\mathcal{I})| - |E(\text{TM}) \setminus \sigma|) \geq |E(\text{TM}) \setminus \sigma| + |\text{dirty}(\mathcal{I})|$. □

The idea is to use MinFas(TM, $k_t - |\text{dirty}(\mathcal{I})|$) for large values of $|\text{dirty}(\mathcal{I})|$, and develop a search tree algorithm, shown in Algorithm 7, that finds an optimal aggregation in time

```
Algorithm 7: OptAggregation2
  Require: $\mathcal{I}$
1 TM $\leftarrow$ a tournament majority graph of $\mathcal{I}$;
2 $O \leftarrow$ BoundedSearchTree2(TM, $unanimity(\mathcal{I})\}$);
3 return $\sigma \in O$ that minimizes $\tau(\sigma, \mathcal{I})$;
```

$O^*((\sqrt{3})^{|\text{dirty}(\mathcal{I})|})$, thus running quickly for small values of $|\text{dirty}(\mathcal{I})|$. The two algorithms will then complement each other and result in an improved time bound for all values of $|\text{dirty}(\mathcal{I})|$.

Algorithm 7 uses ideas from both MinFas and Betzler et al.'s algorithm [20]. As in Betzler et al.'s algorithm [20], it first fixes all the pair orderings in *unanimity*($\mathcal{I}$) (line 2). However, instead of branching on all possible orderings of triples of dirty pairs, it incorporates a tournament majority graph TM into the search (lines 1-2), and branches on ways of breaking C_3's and C_4's in TM in BoundedSearchTree2, shown in Algorithm 8. The branching on small cycles in BoundedSearchTree2 is essentially what was done in BoundedSearchTree1, except that in places that BoundedSearchTree1 was branching on minimal feedback arc sets of a cycle, BoundedSearchTree2 branches on all feedback arc sets (in fact reverses of them, for technical reasons) of the cycle (lines 5 and 7). This change will decrease the running time for small values of d, for which we will be using Algorithm 7.

```
Algorithm 8: BoundedSearchTree2
   Require: $G, L$
 1 if $G$ does not have a $C_3$ then                    /* no cycles remain */
 2     return $\{E(G)\}$;
 3 else if $G$ has a $C_3$ cycle $C$ with $E(C) \cap L \neq \emptyset$ then
 4     if $E(C) \subseteq L$ then return $\emptyset$;                  /* L has a cycle */
 5     else  $P \leftarrow \{\pi \in \text{Total}(V(C)) : \pi$ is consistent with $L\}$;
 6 else if $G$ has a $C_4$ cycle $C$ then
 7     $P \leftarrow \{\pi \in \text{Total}(V(C)) : \pi$ is consistent with $L\}$;
 8 else                                  /* $C_3$'s in $G$ do not have common arcs */
 9     let $C$ be a $C_3$ in $G$;
10     let $e$ be a minimum-weight arc in $E(C)$;
11     $P \leftarrow \{(E(C) \setminus \{e\}) \cup \text{rev}(e)\}$;
12 return $\bigcup_{\pi \in P}$ BoundedSearchTree2$((G - rev(\pi)) + \pi, L \cup \pi)$;
```

Theorem 5.1. *OptAggregation2($\mathcal{I}$) returns an optimal aggregation of $\mathcal{I}$ in time $O^*((\sqrt{3})^d)$, where $d = |\text{dirty}(\mathcal{I})|$.*

Proof. Every relation returned by BoundedSearchTree2 is a total order (lines 1, 2). If every optimal aggregation is returned by BoundedSearchTree2, OptAggregation2 is guaranteed to return an optimal aggregation at line 3. This is true since none of the total orders are eliminated in the branchings of BoundedSearchTree2.

Therefore, it suffices to prove that OptAggregation2($\mathcal{I}$) runs in $O^*((\sqrt{3})^d)$ time. We use $u(L)$ to denote the number of undecided pairs, i.e. $|\{\{a, b\} : a, b \in U, (a, b) \notin L\}|$. Initially $u(L)$ is d. When $u(L)$ becomes zero, the algorithm returns: either G does not have a cycle at that point, causing a return in line 2, or has some C_3's, in which case all the arcs in the C_3's are in L, causing a return in line 4. For non-zero $u(L)$'s, the algorithm returns in one step if G does not have a C_3 (line 2). If G has a C_3 cycle C with two arcs in $E(C) \cap L$, the algorithm branches on the ordering of at most one arc (lines 5 and 12). The value of L will be updated at line 12, causing $u(L)$ to decrease by the number of arcs decided. Similarly, if G has a C_3 cycle C with one arc in $E(C) \cap L$, the algorithm branches on the ordering of two arcs and inserts two arcs in L, reducing $u(L)$ by two in all cases. If G has a C_4 cycle C, then $E(C)$ has either zero or one arc in L. Therefore, either the algorithm branches on 24 cases, reducing $u(L)$ by six in all cases, or branches on 12 cases, reducing $u(L)$ by five in all cases, in line 7. Otherwise, G has to have disjoint C_3's [120] none of which has an arc in L. In this case, the algorithm branches on one case (line 11), reducing $u(L)$ by three.

Consequently, the size of the search tree is bounded by $\alpha^{u(L)}$ for any α that satisfies the following inequalities:

$$\begin{aligned}
&\alpha^0 \geq 1 \\
&\alpha^{u(L)} \geq 1 \times \alpha^{u(L)-1} \\
&\alpha^{u(L)} \geq 3 \times \alpha^{u(L)-2} \\
&\alpha^{u(L)} \geq 12 \times \alpha^{u(L)-5} \\
&\alpha^{u(L)} \geq 24 \times \alpha^{u(L)-6} \\
&\alpha^{u(L)} \geq 1 \times \alpha^{u(L)-3}
\end{aligned}$$

Therefore, the running time of the algorithm is in $O^*((\sqrt{3})^{u(L)})$. Since the initial value of $u(L)$ is d, this time bound is bounded by $O^*((\sqrt{3})^d)$, as required. □

Theorem 5.2. *For any instance $(\mathcal{I}, \frac{k_t}{m})$ of* P-KEMENY RANK AGGREGATION*, an optimal aggregation of $\mathcal{I}$ can be found in $O^*(1.403^{k_t})$ time.*

Proof. Let d denote $|\text{dirty}(\mathcal{I})|$. If $d \geq \frac{2\log_2(1+\sqrt{2})k_t}{\log_2(3)+2\log_2(1+\sqrt{2})}$, then by Lemma 5.1 we can run MinFAS(TM, $k_t - d$) to obtain an optimal aggregation in time

$$O^*((1+\sqrt{2})^{k_t-d}) \leq O^*((1+\sqrt{2})^{(1-\frac{2\log_2(1+\sqrt{2})}{\log_2(3)+2\log_2(1+\sqrt{2})})k_t}) < O^*(1.403^{k_t}).$$

Otherwise, if $d < \frac{2\log_2(1+\sqrt{2})k_t}{\log_2(3)+2\log_2(1+\sqrt{2})}$, then by Theorem 5.1 we can run OptAggregation2($\mathcal{I}$) to find an optimal aggregation in time

$$O^*((\sqrt{3})^d) \leq O^*((\sqrt{3})^{\frac{2\log_2(1+\sqrt{2})k_t}{\log_2(3)+2\log_2(1+\sqrt{2})}}) < O^*(1.403^{k_t}).$$

$\square$

5.2 The $O^*(4^{\frac{k_t}{m}})$-time Enumeration Algorithm

We develop a search tree algorithm, AggregationSearch, that consumes a complete digraph whose arc-weights satisfy the probability and triangle inequality constraints and finds all minimum feedback arc sets of weight at most k of the input graph. Each feedback arc set returned by AggregationSearch is produced in a leaf of its execution tree. We prove an upper bound of 4^k on the number of leaves in the execution tree, yielding an overall running time of $O(4^k \cdot n^\mu)$, where n is the number of vertices in the input graph and μ denotes the exponent of matrix multiplication. The value of k in instances reduced (according to Observation 2.7 in Section 2.3.3) from P-Kemeny Rank Aggregation is $\frac{k_t}{m}$, where k_t is the τ-distance of optimal aggregations from the input votes, and m is the number of input votes. Analogously, there are no more than $4^{\frac{k_t}{m}}$ optimal aggregations, all of which can be found in $O(nm + 4^{\frac{k_t}{m}} \cdot n^\mu)$ time.

We will see in Section 5.2.7 that there are instances with 4^k minimum feedback arc sets. Furthermore, all these instances correspond to P-Kemeny Rank Aggregation instances. Consequently, the upper bounds for the numbers of (locally) minimum feedback arc sets and (locally) optimal aggregations are tight.

5.2.1 A Toy Example

The algorithm AggregationSearch takes the new approach of finding adjacent pairs of locally minimum feedback arc sets. Observation 2.9, which states that all adjacent pairs

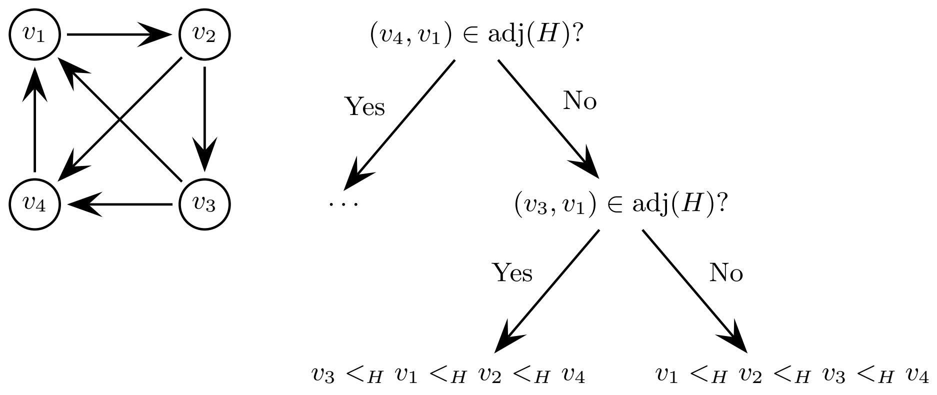
v_1
v_2
v_3
v_4
$(v_4, v_1) \in \text{adj}(H)$?
Yes
No
$\cdots$
$(v_3, v_1) \in \text{adj}(H)$?
Yes
No
$v_3 <_H v_1 <_H v_2 <_H v_4$
$v_1 <_H v_2 <_H v_3 <_H v_4$

Figure 5.1: The first toy example

of a minimum feedback arc set have weight less than or equal to $\frac{1}{2}$, is crucial for the guesses. Indeed, AggregationSearch does not use other properties of minimum feedback arc sets to the point that it actually enumerates all locally minimum feedback arc sets. Locally minimum feedback arc sets are total orders that are only constrained to have their adjacency arcs have weights less than or equal to $\frac{1}{2}$.

To give a sense of how branching on adjacent pairs of a locally minimum feedback arc set $H \in \text{LF}(V, w)$ prunes the search space, we consider the graph shown in Figure 5.1. For clarity, we have omitted arc weights and have drawn only the arcs in $w_{<\frac{1}{2}}$, which must include $\text{adj}(H)$ (i.e., the set of all H-adjacent pairs) for any $H \in \text{LF}(V, w)$.

If $(v_4, v_1) \notin \text{adj}(H)$, v_4 must be ordered last in H, since no other arc of the form $(v_4, *)$ will be left to be in $\text{adj}(H)$. Then, either $(v_3, v_1) \in \text{adj}(H)$, or v_1 must be ordered first in H, since no arc of the form $(*, v_1)$ will remain. In either case, other arcs in $\text{adj}(H)$ are going to be fixed using the same arguments. Different cases and the resulting H's are illustrated in Figure 5.1.

5.2.2 Branching Based on a Feedback Arc Set

A brute-force search for adjacent pairs of an $H \in \text{LF}(V, w)$ can be very inefficient. We will use a minimal feedback arc set F (equivalently, an $F \in \text{Total}(V)$) to ease the search.

The search for the set of arcs $\text{adj}(H) \setminus F$, which we call α, is easy if F has a small weight: the search for α can always be restricted to the set of arcs in $w_{\leq\frac{1}{2}} \setminus F$, which is a super-set of α; a small total weight for F indicates a small number of arcs in $w_{\leq\frac{1}{2}} \setminus F$, since the reverse arcs of $w_{\leq\frac{1}{2}} \setminus F$, each of which has a weight of at least $\frac{1}{2}$, are all in F.

Still, there are possibly many pairs in $w_{\leq\frac{1}{2}} \cap F$ from which to choose the remaining arcs, i.e. $\text{adj}(H) \cap F$. We will show, in Section 5.2.4, that all the arcs in H will be fixed once we figure out those located in a certain region (the region depends on α). The triangle inequalities of the arc weights help, showing that the size of this region is linear in terms of the weight of F, thus a brute-force search in this region is not very costly. The combination of α and the set of arcs of H in the region can be viewed as a concise representation of H in terms of F. We will explain this representation, called the *F-representation of H*, in the next two sections.

We will show in Theorem 5.3 that regardless of the choice of F, AggregationSearch produces all locally minimum feedback arc sets in the leaves of its search tree. The weight w_F of F affects the running time. We prove that the search tree has at most 4^{w_F} leaves and is computed in time $O(4^{w_F} \cdot n^{\mu})$, where μ denotes the exponent of matrix multiplication. As a result, there are at most 4^k locally minimum feedback arc sets in G, where k is the weight of a minimum feedback arc set in G.

5.2.3 A Second Toy Example

We describe the basic idea of our suggested representation for an $H \in \text{LF}(V, w)$ on a small example. The representation is based on an $F \in \text{Total}(V)$. Suppose that the vertices are drawn from left to right in the order of F in Figure 5.2.

In the extreme case that $\alpha = \text{adj}(H) \setminus F$ is $\emptyset$ (i.e., $\text{adj}(H) \subseteq F$), $H = F$: with no $\text{rev}(F)$ arcs in H, H has to order the vertices exactly as F.

Consider a second example where $\alpha = \{(v_5, v_2)\}$. Then, we can be sure that v_1 is ordered first and v_6 is ordered last in H. What we are not sure about is whether either of the vertices v_3 and v_4 is ordered before v_2 and v_5. In fact, everything else will be fixed once we know whether $v_3 <_H v_2$ or $v_2 <_H v_3$, and whether $v_4 <_H v_2$ or $v_2 <_H v_4$. For example, if we figure out that both $v_3 <_H v_2$ and $v_4 <_H v_2$, we must have $v_3 <_H v_4$ since otherwise (v_4, v_3) had to be in $\alpha = \text{adj}(H) \setminus F$ as well. Figure 5.2 shows the decision tree and the resulting H's in each case.

Fortunately, there could not be many vertices in the same situation as v_3 and v_4. By the triangle inequality, the weight of (v_2, v_3) plus the weight of (v_3, v_5), and in general $w(v_2, x)+w(x, v_5)$ for any vertex x satisfying $v_2 <_F x <_F v_5$, is at least the weight of (v_2, v_5). On the other hand, $(v_2, v_5) \in F$ and $w(v_2, v_5) \geq \frac{1}{2}$ since (v_5, v_2) was initially assumed to be

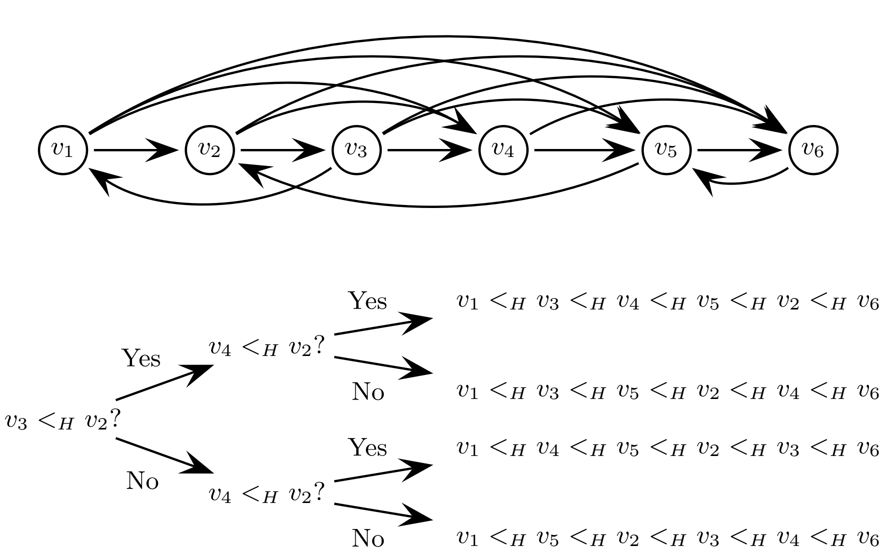
v_1
v_2
v_3
v_4
v_5
v_6
$v_3 <_H v_2$?
Yes
No
$v_4 <_H v_2$?
$v_4 <_H v_2$?
Yes
No
Yes
No
$v_1 <_H v_3 <_H v_4 <_H v_5 <_H v_2 <_H v_6$
$v_1 <_H v_3 <_H v_5 <_H v_2 <_H v_4 <_H v_6$
$v_1 <_H v_4 <_H v_5 <_H v_2 <_H v_3 <_H v_6$
$v_1 <_H v_5 <_H v_2 <_H v_3 <_H v_4 <_H v_6$

Figure 5.2: The second toy example.

in $\alpha \subseteq w_{\leq \frac{1}{2}} \setminus F$. Consequently, the weight of F is at least $\sum_{v_2 <_F x <_F v_5} (w(v_2, x) + w(x, v_5)) \geq \sum_{v_2 <_F x <_F v_5} w(v_2, v_5) \geq |\{v_2 <_F x <_F v_5\}| \cdot \frac{1}{2}$. Therefore, the number of vertices whose relative orders (in H) with respect to v_2 must be determined, as for v_3 and v_4, is at most twice the weight of F.

In the following section, we will see how the bounded number of decisions is generalized to arbitrary α's.

5.2.4 F-Representations

In this section, we define the F-representation of locally minimum feedback arc sets for an arbitrary $F \in \mathrm{Total}(V)$.

The F-representation of $H \in \mathrm{LF}(V, w)$ consists of two parts. The first part, α, is the set $\mathrm{adj}(H) \setminus F$. For a precise definition of the second part, we need to define one term:

Definition 5.2. *An unordered pair $\{x, y\}$ is an F-*internal pair *of $(a, b) \in \mathrm{rev}(F)$ if $x = a$ or $x = b$, and $b <_F y <_F a$.*

We use $\mathrm{IP}_F(e)$ to denote the set of F-internal pairs of $e \in \mathrm{rev}(F)$, and by extension, we use $\mathrm{IP}_F(R)$ for a binary relation $R \subseteq \mathrm{rev}(F)$ to denote $\bigcup_{e \in R} \mathrm{IP}_F(e)$.

Example. Assuming that $v_1 <_F v_2 <_F v_3 <_F v_4 <_F v_5$ for $F \in \mathrm{Total}(\{v_1, \ldots, v_5\})$, $\mathrm{IP}_F((v_5, v_3))$ is $\{\{v_3, v_4\}, \{v_4, v_5\}\}$ and $\mathrm{IP}_F(\{(v_5, v_3), (v_4, v_1)\})$ is $\{\{v_3, v_4\}, \{v_4, v_5\}, \{v_1, v_2\}, \{v_2, v_4\}, \{v_1, v_3\}, \{v_3, v_4\}\}$.

The second part of the F-representation of $H \in \mathrm{LF}(V, w)$ is the restriction of H to $\mathrm{IP}_F(\alpha)$.

Example. Considering the previous example, the restriction of H with $v_1 <_H v_2 <_H v_4 <_H v_5 <_H v_3$ to $\mathrm{IP}_F((v_5, v_3))$ is $H|\mathrm{IP}_F((v_5, v_3)) = \{(v_4, v_3), (v_4, v_5)\}$.

Definition 5.3. *The* F-representation *of* $H \in \mathrm{LF}(V, w)$*, for some* $F \in \mathit{Total}(V)$*, is* (α, δ) *where* $\alpha = \mathrm{adj}(H) \setminus F$ *and* $\delta = H|\mathrm{IP}_F(\alpha)$*.*

A locally minimum feedback arc set can be efficiently reconstructed from its F-representation for an arbitrary $F \in \mathrm{Total}(V)$:

Lemma 5.3. *If* (α, δ) *is the* F*-representation of* $H \in \mathrm{LF}(V, w)$*, then* $H = (\alpha \cup \delta)^+ \cup F \setminus \mathrm{rev}((\alpha \cup \delta)^+)$*.*

Proof. Since $(\alpha \cup \delta)^+ \cup F \setminus \mathrm{rev}((\alpha \cup \delta)^+)$ is a complete relation, it suffices to show that its two subsets $(\alpha \cup \delta)^+$ and $F \setminus \mathrm{rev}((\alpha \cup \delta)^+)$ are in H. The former is true since, by definition, α and δ, and hence $(\alpha \cup \delta)^+$ (because H is transitive), are subsets of H. We prove the latter by showing that $H \setminus F$ is a subset of $(\alpha \cup \delta)^+$. Since H and F are complete relations, $F \setminus H$ will then be a subset of $\mathrm{rev}((\alpha \cup \delta)^+)$, and thus, $F \setminus \mathrm{rev}((\alpha \cup \delta)^+)$ will be a subset of H.

We need to show that every $(x, y) \in H \setminus F$ is in $(\alpha \cup \delta)^+$. The proof is by strong induction: assuming that the claim is true for every $(x', y') \in H \setminus F$ with $y <_F y'$, we prove the claim for (x, y).

Consider drawing the vertices in V in a horizontal line and ordered in the order of F from left to right. Suppose that $z_1 <_H z_2 <_H \ldots <_H z_\ell$, with $z_1 = x$, $z_\ell = y$, $\ell \geq 2$, and $z_i <_{\mathrm{adj}(H)} z_{i+1}$ for all $1 \leq i < \ell$. Figure 5.3 demonstrates an example drawing where $z_7 <_F z_8 <_F y <_F z_5 <_F z_6 <_F x <_F z_3 <_F z_4 <_F z_2$.

Now imagine that we are traversing the vertices starting from z_1, going through the arcs in $\mathrm{adj}(H)$, and finally arriving at z_ℓ. We pass through arcs in $\alpha = \mathrm{adj}(H) \setminus F$ whenever we go from right to left. Note that z_ℓ must be to the left of z_1 in this drawing, since $y <_F x$. Therefore, in order to reach $y = z_\ell$ from $x = z_1$, we need to go through at least one right-to-left edge that ends up at y or some vertex to the left of y (the edge (z_6, z_7) in Fig. 5.3). To be precise, since $(x, y) \in H \setminus F$ there must exist some $1 \leq t < \ell$ such that $(z_t, z_{t+1}) \in \alpha$ with $z_{t+1} \leq_F y <_F z_t$. Observe that when $z_{t+1} \neq y$, $\{y, z_{t+1}\} \in \mathrm{IP}_F(\alpha)$.

We now prove the induction step. Since $z_{t+1} <_F z_t$, every $(x, z_t) \in H \setminus F$ is in $(\alpha \cup \delta)^+$ due to the induction hypothesis.

If $(z_t, z_{t+1}) = (x, y)$, then $(x, y) \in \alpha$, and $(x, y) \in (\alpha \cup \delta)^+$ is trivially true. Otherwise, we demonstrate that $(z_{t+1}, y) \in (\alpha \cup \delta)^+$ if $z_{t+1} \neq y$ and $(x, z_t) \in (\alpha \cup \delta)^+$ if $x \neq z_t$. Together with $(z_t, z_{t+1}) \in \alpha$, these result in $(x, y) \in (\alpha \cup \delta)^+$, as needed to complete the proof.

We first prove that $(z_{t+1}, y) \in (\alpha \cup \delta)^+$ if $z_{t+1} \neq y$. As mentioned above, when $z_{t+1} \neq y$, $\{y, z_{t+1}\}$ is in $\mathrm{IP}_F(\alpha)$. Since H orders z_{t+1} before y, $(z_{t+1}, y) \in H|\mathrm{IP}_F(\alpha) = \delta \subseteq (\alpha \cup \delta)^+$.

Second, considering the relative orders of z_t and x, we prove that $(x, z_t) \in (\alpha \cup \delta)^+$ if $x \neq z_t$.

Case 1: $z_t <_F x$

Since H orders x before z_t, $(x, z_t) \in H \setminus F$ in this case. Therefore, $(x, z_t) \in (\alpha \cup \delta)^+$ by the induction hypothesis.

Case 2: $x <_F z_t$

In this case, $z_{t+1} < x < z_t$. Therefore, $\{x, z_t\} \in \mathrm{IP}_F(\alpha)$. Since H orders x before z_t, $(x, z_t) \in H|\mathrm{IP}_F(\alpha) = \delta \subseteq (\alpha \cup \delta)^+$.

□

5.2.5 The Algorithm

In this section, we describe our search tree algorithm AggregationSearch, shown in Algorithm 9. The algorithm receives a total order $\mathcal{F}$. It then computes every $H \in \mathrm{LF}(V, w)$ through recursive construction of its F-representation (α, δ) in AggregationSearchRec (Algorithm 10).

The *F-length* of an arc $(a, b) \in \mathrm{rev}(F)$, used in AggregationSearchRec, is the number of vertices in $\{y : b <_F y <_F a\}$. We define one more term before describing AggregationSearchRec.

Definition 5.4. *A binary relation R is an* ordering of *a set of unordered pairs P if both $R = R|P$ and $|R| = |P|$.*

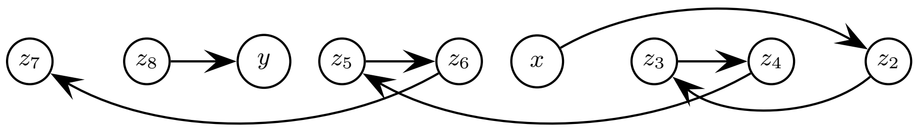

Figure 5.3: An example of the case $x <_H y$ and $y <_F x$, where the vertices are shown in the order of F from left to right and the ordered pairs in $\mathrm{adj}(H)$ are presented as arcs.

By this definition, the second part of an F-representation, i.e. δ, is an ordering of $\mathrm{IP}_F(\alpha)$.

```
Algorithm 9: AggregationSearch
  Require: vertex set $V$, arc weight function $w : V \times V \mapsto R$, and $F \in \mathrm{Total}(V)$
  Assume: $w$ satisfies the triangle inequality and the probability constraint
1 return AggregationSearchRec($V, w, F, \mathrm{rev}(F), \emptyset, \emptyset$);
```

In addition to the input digraph, F, and the partially constructed α and δ, Algorithm AggregationSearchRec (Algorithm 10) uses an auxiliary parameter B. This parameter contains the subset of $\mathrm{rev}(F)$ for which the decision of being in α or not has not yet been made. Initially, B is set to $\mathrm{rev}(F)$. For α to be part of the F-representation of some $H \in \mathrm{LF}(V, w)$, the arcs in α must be contained in $w_{\leq \frac{1}{2}}$, since $\alpha = \mathrm{adj}(H) \setminus F$ is a subset of $\mathrm{adj}(H)$ and $\mathrm{adj}(H)$ must be contained in $w_{\leq \frac{1}{2}}$. Therefore, when $B \cap w_{\leq \frac{1}{2}}$ becomes empty (lines 1-6), the algorithm stops adding arcs to α. By that time, δ is exactly an ordering of $\mathrm{IP}_F(\alpha)$, since for each arc e inserted in α, the algorithm adds all possible orderings of $\mathrm{IP}_F(e)$ to δ. Hence, the algorithm stops adding arcs to δ as well. Due to Lemma 5.3, if the constructed α and δ form an F-representation for an $H \in \mathrm{LF}(V, w)$, H must be equal to $(\alpha \cup \delta)^+ \cup F \setminus \mathrm{rev}((\alpha \cup \delta)^+)$. Therefore, the algorithm checks if this formula produces a locally minimum feedback arc set (line 4). If not, (α, δ) is neither an F-representation for any locally minimum feedback arc set, nor can it be made an F-representation through adding arcs to α and δ.

For each arc (u, v) in $B \cap w_{\leq \frac{1}{2}}$, the algorithm branches on whether or not $(u, v) \in \alpha$. This arc is removed from B once the decision is made. In the branch in which $(u, v) \in \alpha$ (lines 11-19), (u, v) is in $\mathrm{adj}(H)$ and hence even more arcs can be removed from B. Since there is only one vertex that is ordered immediately after u in H (i.e. v) and only one vertex that is ordered immediately before v in H (i.e. u), none of the arcs of the same head or tail with (u, v) can be in $\mathrm{adj}(H)$ in this branch. In particular, none of the arcs in $\bigcup_{x \in P} \{(u, x), (x, v)\}$ can be in $\mathrm{adj}(H) \supseteq \alpha$, thus all these arcs are removed from B.

Algorithm 10: AggregationSearchRec

```
   Require: vertex set $V$, weight function $w$, $F \in \mathrm{Total}(V)$, binary relations
            $B, \alpha, \delta \subseteq V \times V$
1  if $B \cap w_{\leq \frac{1}{2}} = \emptyset$ then
2      $H \leftarrow (\alpha \cup \delta)^{+}$;          /* computing the transitive closure of $\alpha \cup \delta$ */
3      $H \leftarrow H \cup (F \setminus \mathrm{rev}(H))$;
4      if $H \in \mathrm{LF}(V, w)$ then  return $\{H\}$;
5      else return $\emptyset$;
6  end
7  else
8      Select $(u, v) \in B \cap w_{\leq \frac{1}{2}}$ of maximum $F$-length;
9      $B \leftarrow B \setminus \{(u, v)\}$;
10     $\mathrm{LF} \leftarrow \mathrm{AggregationSearchRec}(V, w, F, B, \alpha, \delta)$;
11     $\alpha \leftarrow \alpha \cup \{(u, v)\}$;
12     $P \leftarrow \{x : u <_F x <_F v\}$;
13     $B \leftarrow B \setminus \bigcup_{x \in P} \{(u, x), (x, v)\}$;
14     $L \leftarrow \{x \in P : x <_\delta u \text{ or } x <_\delta v\}$;
15     $R \leftarrow \{x \in P : u <_\delta x \text{ or } v <_\delta x\}$;
16     foreach $L \subseteq A \subseteq P \setminus R$ do
17         $\delta' \leftarrow \delta \cup \bigcup_{x \in A} \{(x, u), (x, v)\} \cup \bigcup_{x \in P \setminus A} \{(u, x), (v, x)\}$;
18         $\mathrm{LF} \leftarrow \mathrm{LF} \cup \mathrm{AggregationSearchRec}(V, w, F, B, \alpha, \delta')$;
19     end
20     return LF;
21 end
```

Further branching occurs on the subset $A = \{x \in P : (x, u) \in H\}$ of vertices in $P = \{x : u <_F x <_F v\}$ (lines 16-19). The orderings of the vertices in P with respect to u and v, determined by A, are essential in determining $\delta = H|\mathrm{IP}_F(\alpha)$ in the F-representation of H.

We do not want to branch over a pair more than once; one strategy is to consider arcs in order of F-length. Without this selection criterion, if in Figure 5.4 (with $B \cap w_{<\frac{1}{2}}$ including (u_1, v_1) and (u_2, v_1) such that $v_1 <_F u_1 <_F u_2$) at line 8 the algorithm selected $(u_1, v_1) \in B \cap w_{<\frac{1}{2}}$ to be excluded from α, then further down the search tree, the algorithm could select $(u_2, v_1) \in \alpha \cap w_{<\frac{1}{2}}$ to be included in α. This would result in branching twice on (u_1, v_1), once for membership in α and once, at line 16, to decide whether $u_1 <_\delta v_1$ or $v_1 <_\delta u_1$.

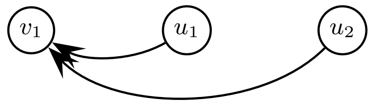

Figure 5.4: Duplicate decisions could be made over the pair $\{u_1, v_1\}$ if the arcs in $B \cap w_{\leq \frac{1}{2}}$ were not selected according to their F-lengths.

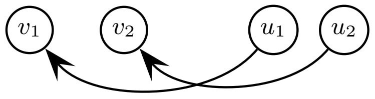

Figure 5.5: Duplicate decisions could be made over the pair $\{u_1, v_2\}$ if A was not constrained to include L and exclude R.

Constraining A to include L and exclude R at line 16 avoids another duplicate branching. Otherwise, the algorithm could decide on relative orderings of vertices in L and R with respect to u and v after the orderings were already fixed in δ. An example situation is illustrated in Figure 5.5 (with $B \cap w_{\leq \frac{1}{2}}$ including (u_1, v_1) and (u_2, v_2) such that $v_1 <_F v_2 <_F u_1 <_F u_2$). Assuming that the algorithm inserts $(u_1, v_1) \in B$ in α at line 11, it needs to decide whether or not to include v_2 in A (a decision on the ordering of $\{u_1, v_2\}$) at line 16. If A was not constrained to include L and exclude R, the algorithm had to decide whether or not to include u_1 in A (a second decision on the ordering of $\{u_1, v_2\}$) if $(u_2, v_2) \in B$ is inserted in α later in the search. The constraint prohibits the algorithm from the second decision.

Removal of the same-head and same-tail arcs from B (line 13), ordering the arcs in B in their F-lengths (line 8), and constraining A to include L and exclude R (line 16) all result in less branching.

5.2.6 Proofs

We are now ready to prove the main theorem. In the proof, we use $\mathrm{LF}_{(F,B,\alpha,\delta)}$ to denote $\{H \in \mathrm{LF}(V, w) : \alpha \subseteq \mathrm{adj}(H) \setminus F \subseteq \alpha \cup B$, and $\delta \subseteq H\}$. Notice that $\mathrm{LF}(V, w) = \mathrm{LF}_{(F,\mathrm{rev}(F),\emptyset,\emptyset)}$ for any $F \in \mathrm{Total}(V)$.

Theorem 5.3. *Given a complete digraph on a vertex set V and arc weight function $w : V \times V \mapsto R$ and $F \in$ Total(V), AggregationSearch(V, w, F) returns* $\mathrm{LF}(V, w)$ *in time* $O(|V|^{\mu} \cdot$

4^{w_F}), *where* $\mu < 2.3727$ *[137] denotes the exponent of matrix multiplication. Furthermore,* $|\mathrm{LF}(V, w)| \leq 4^{w_F}$.

Proof. Every returned binary relation H has satisfied the condition $H \in \mathrm{LF}(V, w)$ at line 4 of AggregationSearchRec. Therefore, the returned binary relations are all in $\mathrm{LF}(V, w)$.

We need to prove that every $H \in \mathrm{LF}(V, w)$ is returned by AggregationSearch(V, w, F). We show that AggregationSearchRec$(V, w, F, B, \alpha, \delta)$ returns every $H \in \mathrm{LF}_{(F,B,\alpha,\delta)}$ if δ is an ordering of $\mathrm{IP}_F(\alpha)$. Since $\mathrm{LF}(V, w) = \mathrm{LF}_{(F,\mathrm{rev}(F),\emptyset,\emptyset)}$ and $\emptyset$ is an ordering of $\mathrm{IP}_F(\emptyset)$, this is all we need to prove.

We use strong induction on the cardinality of $B \cap w_{\leq\frac{1}{2}}$, considering only instances in which δ is an ordering of $\mathrm{IP}_F(\alpha)$. When $|B \cap w_{\leq\frac{1}{2}}| = 0$, the algorithm returns $\{H \in \mathrm{LF}(V, w) : H = (F - \mathrm{rev}((\alpha \cup \delta)^+)) \cup (\alpha \cup \delta)^+\}$ at line 4. By the definition of $\mathrm{LF}(V, w)$, $\mathrm{adj}(H)$ is completely disjoint from B in this case for any $H \in \mathrm{LF}_{(F,B,\alpha,\delta)}$; therefore, $\mathrm{adj}(H) \setminus F = \alpha$ for any $H \in \mathrm{LF}_{(F,B,\alpha,\delta)}$. Also, since δ is assumed to be an ordering of $\mathrm{IP}_F(\alpha)$, $\delta = H|\mathrm{IP}_F(\alpha)$ for any $H \in \mathrm{LF}(V, w)$ that includes δ. Therefore, $\mathrm{LF}_{(F,B,\alpha,\delta)} = \{H \in \mathrm{LF}(V, w) : \mathrm{adj}(H) \setminus F = \alpha$, and $\delta = H|\mathrm{IP}_F(\alpha)\}$, which is equal to the set that is returned (Lemma 5.3), as required.

For non-empty $B \cap w_{\leq\frac{1}{2}}$'s, we would like to apply the induction hypothesis to the recursive calls at lines 10 and 18. That is, we need to make sure that δ is an ordering of $\mathrm{IP}_F(\alpha)$ for all the values of α and δ passed to the recursive calls. The condition holds for the recursive call at line 10, since α and δ are not changed, and the assumption holds for the current α and δ. For recursive calls at line 18, δ' is a union of δ and an ordering of the pairs in $\mathrm{IP}_F((u, v))$. Therefore, assuming that δ is an ordering of $\mathrm{IP}_F(\alpha)$, δ' will be an ordering of $\mathrm{IP}_F(\alpha \cup \{u, v\})$.

Consequently, the algorithm returns $\mathrm{LF}_{(F,B\setminus\{(u,v)\},\alpha,\delta)} \cup \bigcup_{L\subseteq A\subseteq P\setminus R} \mathrm{LF}_{(F,B\setminus(\{(u,v)\}\cup B'),\alpha\cup\{(u,v)\},\delta')}$ for $B' = \bigcup_{x\in P}\{(u, x), (x, v)\}$, $P = \{x : u <_F x <_F v\}$, and $\delta' = \delta \cup \bigcup_{x\in A}\{(x, u), (x, v)\} \cup \bigcup_{x\in P\setminus A}\{(u, x), (v, x)\}$. The proof of correctness is completed if every $H \in \mathrm{LF}(V, w)$ is in this set.

Case 1: $(u, v) \notin \mathrm{adj}(H)$**.** In this case, $\mathrm{adj}(H) \setminus F$ is in $\alpha \cup (B \setminus \{(u, v)\})$. Therefore, $H \in \mathrm{LF}_{(F,B\setminus\{(u,v)\},\alpha,\delta)}$.

Case 2: $(u, v) \in \mathrm{adj}(H)$**.** In this case, $\alpha\cup\{(u, v)\}$ is a subset of $\mathrm{adj}(H)$. Also, none of the pairs in $\mathrm{adj}(H)\setminus\{(u, v)\}$ have common heads or tails with (u, v), thus $\mathrm{adj}(H)\setminus\{(u, v)\}$ has no intersection with B'. Therefore, $\mathrm{adj}(H) \setminus F$ is in $\alpha \cup \{(u, v)\} \cup (B \setminus (\{(u, v)\} \cup B'))$. Furthermore, $\delta' \subseteq H$ when A is set to $\{z \in P : (z, u) \in H\}$. Note that this is an acceptable value for A, i.e. $L \subseteq A \subseteq P \setminus R$ (line 16), since otherwise $\delta \not\subseteq H$, which contradicts the definition of H. Consequently, $H \in \mathrm{LF}_{(F, B\setminus(\{(u,v)\}\cup B'), \alpha\cup\{(u,v)\}, \delta')}$.

To see that the running time bound is met, we associate each node in the search tree with the cost of steps 8-10 or steps 11-18 performed just before the node was created (if any) plus the cost of steps 1-6 performed at the time the node is being executed. The dominant part is the computation of the transitive closure $(\alpha \cup \delta)^+$ using matrix multiplication at line 2. The time for a node is thus in $O(|V|^\mu)$, yielding $O(|V|^\mu \cdot 4^{w_F})$ time overall.

In each internal node of the search tree, line 18 is executed at least once (when A is set to L). Consequently, each internal node has at least two children, and therefore, the number of nodes in the tree is at most twice the number of leaves.

We claim that AggregationSearchRec$(V, w, F, B, \alpha, \delta)$ produces a search tree with no more than $4^{w_{\mathrm{rev}(B)}}$ leaves, if $B \cup \delta$ includes an ordering of $\mathrm{IP}_F(B \cap w_{\le \frac{1}{2}})$. Since $B = \mathrm{rev}(F)$ in the call to AggregationSearchRec$(V, w, F, \mathrm{rev}(F), \emptyset, \emptyset)$, it is a total order, thus includes orderings of arbitrary sets of pairs, including $\mathrm{IP}_F(B \cap w_{\le \frac{1}{2}})$. Therefore, the claim gives an upper bound of $4^{w_{\mathrm{rev}(B)}} = 4^{w_F}$ on the number of leaves, as required.

We use strong induction on the cardinality of $B \cap w_{\le \frac{1}{2}}$, considering only instances in which $B \cup \delta$ is an ordering of $\mathrm{IP}_F(B \cap w_{\le \frac{1}{2}})$. When $|B \cap w_{\le \frac{1}{2}}| = 0$, the algorithm terminates at line 4 or 5 with no recursive calls made. Therefore, only 1 leaf is produced, which does not exceed $4^{w_{\mathrm{rev}(B)}}$, as required.

For non-empty $B \cap w_{<\frac{1}{2}}$'s, we would like to apply the induction hypothesis to the recursive calls at lines 10 and 18. That is, we need to make sure that $B \cup \delta$ is an ordering of $\mathrm{IP}_F(B \cap w_{<\frac{1}{2}})$ for all the values of B and δ used in the recursive calls. This is true for the call at line 10: the F-length of the arc (u, v) is at least as large as the F-lengths of the arcs in $(B \setminus \{(u,v)\}) \cap w_{<\frac{1}{2}}$. By the definition of IP, the F-length of any arc ordering a pair in $\mathrm{IP}_F((B \setminus \{(u,v)\}) \cap w_{<\frac{1}{2}})$ is smaller than the F-length of an arc in $(B \setminus \{(u,v)\}) \cap w_{<\frac{1}{2}}$. Therefore, (u, v) is not an ordering of any of the pairs in $\mathrm{IP}_F((B \setminus \{(u,v)\}) \cap w_{<\frac{1}{2}})$, and hence, $(B \setminus \{(u,v)\}) \cup \delta$ still includes an ordering of $\mathrm{IP}_F((B \setminus \{(u,v)\}) \cap w_{<\frac{1}{2}})$. For the recursive calls at line 18, the removal of $\bigcup_{x\in P}\{(u,x),(x,v)\}$ from B does not cause a problem, since for any arc e removed at line 13, either e or $\mathrm{rev}(e)$ is added to δ at line 17, thus the new $B \cup \delta$ includes an ordering of any set of pairs for which $(B \setminus \{(u,v)\}) \cup \delta$ of line 10 included an ordering.

Consequently, by the induction hypothesis, the algorithm produces at most $4^{w_{\mathrm{rev}(B\setminus\{(u,v)\})}} = 4^{w_{\mathrm{rev}(B)} - w(v,u)}$ leaves at line 10 and $4^{w_{\mathrm{rev}(B\setminus(\{(u,v)\}\cup B'))}}$ leaves in each of the recursive calls at line 18 (for $B' = \bigcup_{x\in P}\{(u,x),(x,v)\}$ and $P = \{x : u <_F x <_F v\}$), for a total of at most $2^{|P\setminus(L\cup R)|} \cdot 4^{w_{\mathrm{rev}(B\setminus(\{(u,v)\}\cup B'))}}$ leaves produced at line 18.

By the probability constraint, $w(v,u) = 1 - w(u,v)$. Therefore, $(u,v) \in w_{<\frac{1}{2}}$ leads to $w(v,u) \geq \frac{1}{2}$, proving the upper bound of $4^{w_{\text{rev}(B)}-\frac{1}{2}}$ on $4^{w_{\text{rev}(B)}-w(v,u)}$. On the other hand, since $B \cup \delta$ includes an ordering of $\text{IP}_F(B \cap w_{<\frac{1}{2}})$, $B \cup \delta$ includes an ordering of $\text{IP}_F(\{(u,v\})$. By definitions of L and R at lines 14 and 15, $\bigcup_{x \in L \cup R}\{\{u,x\},\{x,v\}\}$ are the only pairs for which δ can possibly be an ordering. Hence, B includes an ordering of $P' = \bigcup_{x \in P \setminus (L \cup R)}\{\{u,x\},\{x,v\}\}$. By the definition of P and the fact that $B \subseteq \text{rev}(F)$, $B|P' = \bigcup_{x \in P \setminus (L \cup R)}\{(u,x),(x,v)\}$. Therefore, $w_{\text{rev}(B \setminus (\{(u,v)\} \cup B'))} \leq w_{\text{rev}(B \setminus \{(u,v)\})} - w_{\text{rev}(B|P')} = w_{\text{rev}(B)} - w(v,u) - \sum_{x \in P \setminus (L \cup R)} w(x,u) + w(v,x)$. The triangle inequality helps here, proving that $\sum_{x \in P \setminus (L \cup R)} w(x,u) + w(v,x) \geq \sum_{x \in P \setminus (L \cup R)} w(v,u) = |P \setminus (L \cup R)| \cdot w(v,u)$. Since $w(v,u) \geq \frac{1}{2}$, $2^{|P \setminus (L \cup R)|} \cdot 4^{w_{\text{rev}(B \setminus \{(u,v)\} - B')}}$ is at most $2^{|P \setminus (L \cup R)|} \cdot 4^{w_{\text{rev}(B)} - \frac{|P \setminus (L \cup R)|+1}{2}}$.

Overall, there will be at most

$$4^{w_{\text{rev}(B)}-\frac{1}{2}} + 2^{|P \setminus (L \cup R)|} \cdot 4^{w_{\text{rev}(B)} - \frac{|P \setminus (L \cup R)|+1}{2}} = 4^{w_{\text{rev}(B)}}$$

leaves produced, as needed to prove the claim. □

As a consequence of Observation 2.12, P-Kemeny Rank Aggregation instances have at most $4^{\frac{k_t}{m}}$ locally optimal aggregations.

Corollary 5.1. *Given a multi-set $\mathcal{I}$ of m total orders in Total(U) and a total order σ at τ-distance k_σ of $\mathcal{I}$, the set of all locally optimal aggregations for $\mathcal{I}$ can be found in time $O(nm + 4^{\frac{k_\sigma}{m}} \cdot n^\mu)$, where μ denotes the exponent of matrix multiplication. Furthermore, $\mathcal{I}$ has at most $4^{\frac{k_t}{m}}$ locally optimal aggregations, where k_t denotes the minimum τ-distance from $\mathcal{I}$.*

5.2.7 The Running Time is Tight

Theorem 5.3's upper bound on $|\text{LF}(V,w)|$ is tight for some instances. Although $\text{MF}(V,w)$ is generally a (small) subset of $\text{LF}(V,w)$, the two sets are equal in the instances constructed for Theorem 5.4:

Theorem 5.4. *For any set $V = \{v_1, v_2, \ldots, v_n\}$ of even cardinality, there exists a weight function w over $V \times V$ that satisfies the triangle inequality and the probability constraints such that $|\text{MF}(V,w)| = 4^k$, where k denotes the weight of a minimum feedback arc set in $\text{MF}(V,w)$.*

Proof. We assign the weight $w(v_i, v_j)$ to each arc (v_i, v_j) in the loopless complete digraph G over the vertex set V:

$$w(v_i, v_j) = \begin{cases} 0 & i+1 < j \text{ or } (i+1 = j \text{ and } i \text{ is even}) \\ \frac{1}{2} & i+1 = j \text{ and } i \text{ is odd} \\ 1 - w(v_j, v_i) & \text{otherwise} \end{cases}$$

It is easy to see that all weight-0 arcs must be included in any minimum feedback arc set for G. It remains to determine the ordering of the pairs $\{v_1, v_2\}$, $\{v_3, v_4\}$, and so on. The weights of all the options are equal. Therefore, any total order that includes all weight-0 arcs and an ordering of $\{\{v_1, v_2\}, \{v_3, v_4\}, \ldots \{v_{n-1}, v_n\}\}$ is a minimum feedback arc set for G. There are $2^{\frac{n}{2}}$ such total orders, each of weight $k = \frac{n}{4}$. Therefore, the cardinality of $\text{MF}(V, w)$ is $2^{2k} = 4^k$ for this instance. □

There are P-Kemeny Rank Aggregation instances that reduce to the instances in the proof of Theorem 5.4. Thus, the lower bound can be carried over to the number of optimal aggregations:

Theorem 5.5. *For any even number m, there exists a multi-set $\mathcal{I}$ of m total orders that has $4^{\frac{k_t}{m}}$ optimal aggregations, where k_t denotes the τ-distance of an optimal aggregation from $\mathcal{I}$.*

Proof. For any set $U = \{u_1, \ldots, u_n\}$ of even cardinality, the $\mathcal{I}$ instance which includes $\frac{m}{2}$ copies of the total order $\{\pi \in \text{Total}(U) : u_1 <_\pi u_2 <_\pi u_3 <_\pi u_4 \ldots <_\pi u_{n-1} <_\pi u_n\}$ and $\frac{m}{2}$ copies of the total order $\{\pi \in \text{Total}(U) : u_2 <_\pi u_1 <_\pi u_4 <_\pi u_3 \ldots u_n <_\pi u_{n-1}\}$ reduces to Theorem 5.4's instance constructed for $V = U$. Therefore, $\mathcal{I}$ has $4^{\frac{k_t}{m}}$ optimal aggregations, as required. □

5.3 Kernels

In this section, we describe our kernels for WDFAS instances that satisfy triangle inequality and probability constraint, which correspond to partial kernels for P-Kemeny Rank Aggregation instances. Both kernels that we provide are based on the WDFAS variant of the Extended Condorcet Criterion (Observation 2.10 in Section 2.3.4), saying that when there exists a subset $V' \subseteq V$ satisfying $(V' \times (V \setminus V')) \subseteq w_{<\frac{1}{2}}$, then the set of arcs in $(V' \times (V \setminus V'))$ is included in every minimum feedback arc set.

5.3.1 The Enumerative Kernel

We start with a $4k$-vertex enumerative kernel. We do not need the full power of Observation 2.10 for this kernel. Rule 1 formalizes a special case of Observation 2.10 in which V' is the set of $w_{<\frac{1}{2}}$-in-neighbours of a vertex or $V \setminus V'$ is the set of $w_{<\frac{1}{2}}$-out-neighbours of a vertex; note that such a vertex cannot be in a 3-cycle in $w_{\leq\frac{1}{2}}$.

Rule 1: If a vertex v does not belong to a 3-cycle in $w_{\leq\frac{1}{2}}$, remove its $w_{\neq\frac{1}{2}}$-arcs and the arcs between its $w_{\leq\frac{1}{2}}$-in-neighbours and its $w_{\leq\frac{1}{2}}$-out-neighbours.

Lemma 5.4. *Rule 1 is sound.*

Proof. It suffices to show that the relative orderings of v and its $w_{\neq\frac{1}{2}}$-neighbours, as well as relative orderings of v's $w_{\leq\frac{1}{2}}$-in-neighbours with respect to v's $w_{\leq\frac{1}{2}}$-out-neighbours, are the same in all $F \in \mathrm{MF}(V, w)$.

Consider a vertex v that does not belong to a 3-cycle in $w_{\leq\frac{1}{2}}$. Let L denote the set of $w_{<\frac{1}{2}}$-in-neighbours of v and R denote the set of $w_{<\frac{1}{2}}$-out-neighbours of v. The definition of v ensures that there is no arc in $w_{\leq\frac{1}{2}}$ from R to L, thus all the arcs from L to R (the reverse of the arcs from R to L) are in $w_{<\frac{1}{2}}$ by the probability constraint.

Consider the set of vertices outside of $L \cup R \cup \{v\}$. By definitions of L and R, $(v, x), (x, v) \in w_{\leq\frac{1}{2}}$ for every vertex x in this set. If there are two (or more) vertices x and y outside of $L \cup R \cup \{v\}$, then v and x and y will form a 3-cycle in $w_{\leq\frac{1}{2}}$ (Fig. 5.6).

If all the vertices are in $L \cup R \cup \{v\}$, we can apply Observation 2.10 twice to show that $(L \times (\{v\} \cup R)) \subseteq F$ and $((L \cup \{v\}) \times R) \subseteq F$ for all $F \in \mathrm{MF}(V, w)$, completing the proof for this case.

If instead there is a vertex $x \notin L \cup R \cup \{v\}$, the absence of a 3-cycle containing v implies that $L \times \{x\} \subseteq w_{<\frac{1}{2}}$ and $\{x\} \times R \subseteq w_{<\frac{1}{2}}$. Consequently, in this case Observation 2.10 can be applied to show that $(L \times (\{v, x\} \cup R)) \in F$ and $((L \cup \{v, x\}) \times R) \in F$ for all $F \in \mathrm{MF}(V, w)$. □

The proof of Lemma 5.4 specifies the restrictions of (all) minimum feedback arc sets to the set of removed arcs:

Lemma 5.5. *For every $F \in \mathrm{MF}(V, w)$ and for α the set of arcs removed from an instance after multiple applications of Rule 1, $F|\alpha \subseteq w_{<\frac{1}{2}}$.*

Rule 2: Remove an isolated vertex.

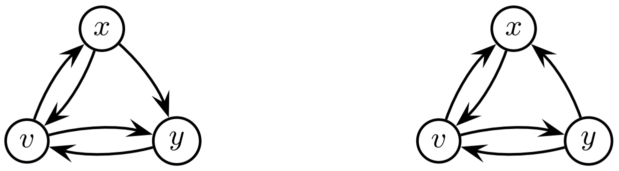

Figure 5.6: 3-cycles produced if a vertex has two neighbours in $w_{=\frac{1}{2}}$

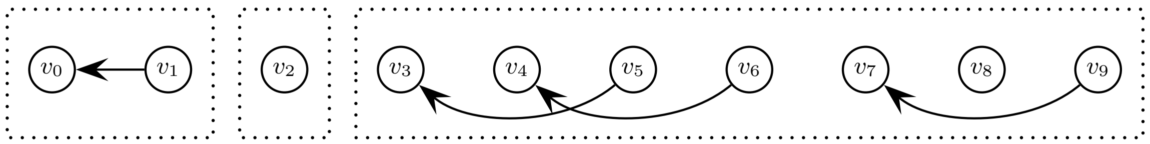

Figure 5.7: Strongly connected components formed after exhaustive application of Rule 1

Example 5.1. In the complete digraph depicted in Figure 5.7, assume that the arcs drawn are the subset of arcs in $w_{\geq\frac{1}{2}}$ that are ordered from right to left. The vertex v_2 does not belong to any 3-cycles in $w_{\leq\frac{1}{2}}$, since the reverse arcs of such a 3-cycle (all in $w_{\geq\frac{1}{2}}$) must include an arc going from right to left, i.e. one of the arcs drawn in the figure. Therefore, Rule 1 removes $w_{\neq\frac{1}{2}}$-arcs incident to v_2 and the arcs between the sets $\{v_0, v_1\}$ and $\{v_3, v_4 \dots, v_9\}$. None of the arcs incident to v_2 are in $w_{=\frac{1}{2}}$. Hence, Rule 1 isolates v_2, allowing Rule 2 to remove it. The dotted lines in the figure specify the resulting strongly connected components after an application of Rule 1. Notice that none of the arcs between pairs of vertices in $C = \{v_3, v_4, \dots, v_9\}$ are removed by Rule 1, since each of the vertices in C is included in a 3-cycle in $w_{\leq\frac{1}{2}}$. Therefore, the complete digraph over C will be untouched by Rules 1 and 2.

The following terms are used to specify the structure in the reduced instances. We define the F*-span* of an arc (u, z) to refer to the set of vertices ordered between u and z in a total order F:

Definition 5.5. *A vertex $v \in V$ is in the F-*span *of an arc $(u, z) \in F$ for a total order $F \in$ Total(V) if $u <_F v <_F z$.*

A relaxation of the F-span concept, involving a set of arcs rather than a set of vertices, will also be useful in our descriptions:

Definition 5.6. *An arc (x, y) is in the F-*span *of an arc $(u, z) \in F$ for a total order $F \in Total(V)$ if $x = u$ and y is in the F-span of (u, z), $y = z$ and x is in the F-span of (u, z), or both x and y are in the F-span of (u, z).*

Example 5.2. The F-span of $(1, 3)$ for the total order $F \in \mathcal{T}_U$, $U = \{1, 2, 3, 4\}$, satisfying $1 <_F 2 <_F 3 <_F 4$ is the set of vertices and arcs $\{2, (1, 2), (2, 3)\}$.

Lemma 5.6. *After exhaustive applications of Rules 1 and 2 on a complete digraph, each of the vertices in the reduced instance is either an endpoint or in the F-span of an arc in $F \cap w_{\geq \frac{1}{2}}$, for any total order F.*

Proof. Suppose instead that the reduced instance contains a vertex v that is neither in endpoints$(F \cap w_{\geq \frac{1}{2}})$ nor in the F-span of an arc in $F \cap w_{\geq \frac{1}{2}}$.

Since any 3-cycle must contain at least one arc in any total order, any 3-cycle in $w_{\geq \frac{1}{2}}$ contains an arc in F. Thus v does not belong to such a 3-cycle, as either an arc incident to v would be in $F \cap w_{> \frac{1}{2}}$, or v would be in the F-span of an arc in $F \cap w_{> \frac{1}{2}}$, a contradiction.

Since Rules 1 and 2 remove arcs between but not within strongly connected components, no reduced instance of a complete digraph will have an incomplete digraph as a strongly connected component. Consequently, any set of vertices that forms a cycle in $w_{\leq \frac{1}{2}}$ will also form a 3-cycle in $w_{\geq \frac{1}{2}}$; we conclude that v does not belong to a 3-cycle in $w_{\leq \frac{1}{2}}$.

In a reduced instance, applying Rule 1 does not result in the removal of any arcs, and hence all the arcs incident to v have to be in $w_{= \frac{1}{2}}$. The inapplicability of Rule 2 implies that v has at least one neighbour. Since each strongly connected component is a complete digraph, v has to be incident to at least two arcs, e and rev(e) (both in $w_{= \frac{1}{2}} \subseteq w_{\geq \frac{1}{2}}$), one of which must be in $F \cap w_{\geq \frac{1}{2}}$. The resulting contradiction proves the lemma. □

Lemma 5.7. *A reduced instance of* WDFAS*, formed by exhaustive application of Rules 1 and 2, has at most $4k$ vertices, where k is the minimum weight of a feedback arc set in the original instance.*

Proof. For F a minimum feedback arc set for the original instance, we associate with each vertex v a set $g(v)$ of one or two arcs in F, of total weight at least $\frac{1}{2}$. We obtain a lower bound on the weight of F by correlating the sets with a partitioning of the arcs.

Let F' be the subset $F \cap w_{\geq \frac{1}{2}}$ and let V' be the set of vertices in the reduced instance. We define $g(v)$ for each $v \in V'$ as follows; g is well-defined, since by Lemma 5.6 every vertex in V' is an endpoint or in the F-span of an arc in F'.

Case 1: If v is an endpoint of some arc $e \in F'$, then $g(v) = \{e\}$.

Case 2: If v is in the F-span of some $(u, z) \in F'$, then $g(v) = \{(u, v), (v, z)\}$.

The weight of $g(v)$ is at least $\frac{1}{2}$ for each v: this is trivially correct for Case 1, since $e \in w_{\geq \frac{1}{2}}$. By the triangle inequality, $w(u, v) + w(v, z) \geq w(u, z)$, thus the total weight of the arcs in $g(v)$ in Case 2 is also at least $\frac{1}{2}$.

To see that each arc in $F_g = \bigcup_{v \in V'} g(v)$ is included in at most two g's, we first observe that each arc in F' is assigned to at most two vertices (its two endpoints, Case 1). Any other arc has exactly one endpoint in endpoints(F'); the endpoint $v \notin F'$ is the endpoint of exactly two arcs, both appearing in the same g set in Case 2.

Consequently,

$$w_{F_g} \geq \frac{1}{2} \cdot \sum_{v \in V'} w_{g(v)} \geq \frac{1}{2} \cdot \frac{1}{2} \cdot |V'|,$$

and thus,

$$4w_F \geq |V'|.$$

□

The function g is important only in deriving the bound; the assignments of arcs to vertices are not actually used in the formation of the kernel.

Theorem 5.6. WDFAS *restricted to complete digraphs whose arc-weights satisfy probability and triangle inequality constraints admits a* $4k$ *vertex kernel; the kernel is enumerative and can be computed in* $O(n^2)$ *time.*

Proof. By Lemma 5.7, we know that there are no more than $4k$ vertices in the digraph G' left after exhaustive application of Rules 1 and 2 to an input digraph G. One can make G' a complete digraph by adding back those arcs removed from G that were between the vertices in G'. As Lemma 5.5 gives a bijection between minimum feedback arc sets in G and minimum feedback arc sets in G', G' is an enumerative kernel.

Exhaustive application of Rules 1 and 2 has an effect similar to computing the strongly connected components of $w_{<\frac{1}{2}}$ and can be done in $O(n^2)$ time. □

Corollary 5.2. P-Kemeny Rank Aggregation *admits an enumerative partial kernel with at most* $\frac{4k_t}{m}$ *candidates.*

Proof. Since the weights of the arcs are not changed by the reductions and the reduced instance is still a complete digraph, the reduced instance still corresponds to a P-Kemeny Rank Aggregation instance. For every optimal aggregation σ of a P-Kemeny Rank Aggregation instance $(\mathcal{I}, \frac{k_t}{m})$, $\mathrm{rev}(\sigma)$ is a minimum-weight feedback arc set of weight $\frac{k_t}{m}$ in the corresponding WDFAS instance (Observation 2.7 in Section 2.3.3). Therefore, Theorem 5.6's bound in terms of k proves that the number of vertices (corresponding to candidates) in the kernel is at most $\frac{4k_t}{m}$. □

5.3.2 Refining the Kernel

We slightly modify Rule 1 so that the number of vertices is now at most $2k + p$, where p is the number of strongly connected components after exhaustive application of the modified rule. In order to force p to be small with respect to k, we introduce an additional rule to remove small strongly connected components.

Our first rule, also mentioned by Betzler et al. [18], is the implementation of Observation 2.10 (in Section 2.3.4), which is the WDFAS variant of the Extended Condorcet Criterion:

Rule 1′: If there exists a subset of vertices $V' \subset V$ such that $(V' \times (V \setminus V')) \subseteq w_{<\frac{1}{2}}$, then remove the arcs between V' and $V \setminus V'$ and reduce k by the total weight of arcs from $V \setminus V'$ to V'.

The proof of Lemma 5.4 follows immediately from Observation 2.10.

Lemma 5.8. *Rule 1′ is sound.*

Rule 1′ slightly differs from Rule 1. It might remove some of the arcs to/from vertices not in a 3-cycle in $w_{\leq\frac{1}{2}}$. As a result, the strongly connected components left after applications of Rule 1′ are subsets of the strongly connected components left after applications of Rule 1. The resulting strongly connected components are still guaranteed to be complete digraphs.

Example 5.3. Consider the graph in Example 5.1. Like Rule 1, Rule 1′ isolates v_2. The difference is that Rule 1′ removes the arcs from vertices in $\{v_0, v_1 \ldots, v_6\}$ to vertices in $\{v_7, v_8, v_9\}$. The resulting strongly connected components for both the rules are shown by dotted lines in Figures 5.7 and 5.8.

Rule 1′ results in the following refined bound:

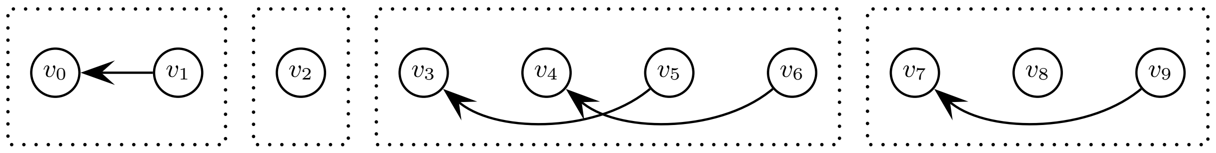

Figure 5.8: The strongly connected components formed after exhaustive application of Rule $1'$

Lemma 5.9. *A reduced instance of* WDFAS*, formed by exhaustive application of Rules $1'$ and 2, has at most $2k + p$ vertices, where k is the minimum weight of a feedback arc set F in the original instance and p is the number of strongly connected components in the reduced instance.*

Proof. We set $F' = \{e \in F \cap w_{\geq \frac{1}{2}} :$ no arc $e' \in F \cap w_{\geq \frac{1}{2}}$ is in the F-span of $e\}$. We associate a set $g(v)$ to each $v \in V'$ as follows, where V' is the set of vertices in the reduced instance:

Case 1: If v is the tail of some arc $(u, v) \in F'$, then $g(v) = \{(u, v)\}$.

Case 2: If v is in the F-span of some $(u, z) \in F'$, then $g(v) = \{(u, v), (v, z)\}$.

Case 3: If v is the head of some arc $(v, u) \in F'$, then $g(v) = \emptyset$.

Except for one vertex in each strongly connected component (the one that is ordered first in F), all vertices are either the tail or in the F-span of some arc in F'. Suppose that a vertex v is not the tail or in the F-span of an arc in F'. Let L be the set of vertices ordered before v in F. Then, it is easy to see that every arc (in the original instance) from L to the rest of the vertices, including v, is in $w_{<\frac{1}{2}}$. Therefore, Rule $1'$ removes all the arcs between L and v. Consequently, v is ordered first (by F) in its strongly connected component in the reduced instance.

Therefore, the number of vertices assigned to Case 3 is p (one for each strongly connected component). To complete the proof, it will suffice to show that all the arcs in g are distinct, since then

$$w_F \geq \sum_{v \in V'} w_{g(v)} \geq \frac{1}{2} \cdot (|V'| - p),$$

thus,

$$2w_F + p \geq |V'|,$$

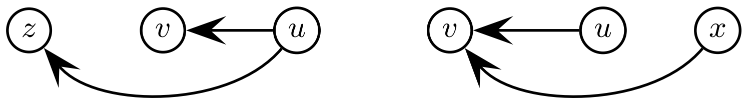

Figure 5.9: Situations occurring if (u, v) were to be assigned both in Case 1 and in Case 2

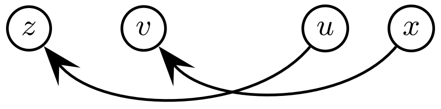

Figure 5.10: The situation occurring if (u, v) were to be assigned twice in Case 2

as required.

The tails of the arcs assigned in Case 1 are all distinct, hence the arcs themselves are also distinct. If an arc (u, v) were to be assigned once in Case 1 and also in Case 2, then either v would be in the F-span of an arc $(u, z) \in F'$ or u would be in the F-span of an arc $(x, v) \in F'$, as depicted in Figure 5.9. However, in each case (u, v) is in the F-span of the other arc in the figure, thus indicating that such an arc is not in F'. Finally we show that no arc can be associated with two different vertices in Case 2; this could only occur if u were in the F-span of an arc (x, v) and v were in the F-span of an arc (u, z), as depicted in (Figure 5.10). However, v is the tail of $(x, v) \in F'$ in this situation, and falls into Case 1, which is a contradiction. □

Our next rule is based on the fact that each strongly connected component can be solved independently:

Rule 3: Solve WDFAS for a strongly connected component of at most $\frac{2}{\epsilon}$ vertices, remove the component, and decrease k accordingly.

Observation 5.2. *Rule 3 is sound.*

Proof. Suppose that (G, k) is an instance of WDFAS and it reduces to (G', k') after an application of Rule 3. To be more precise, suppose that G is a disjoint union of complete digraphs, C is a strongly connected component in G (thus, also a complete digraph), and F_C is a minimum weight feedback arc set for C. Then, G has a feedback arc set of weight k if and only if $G' = (V(G) - V(C), E(G) - E(C))$ has a feedback arc set of weight

$k' = k - w_{F_C}$. Therefore, regardless of the bound $\frac{2}{\epsilon}$ on the number of vertices in C, Rule 3 is sound. □

When Rules 1′ and 2 are the only rules applied, there is a bijection between minimum feedback arc sets in the original instance and minimum feedback arc sets in the reduced instance. The bijection can be lost after an application of Rule 3, since the possibly multiple solutions for a strongly connected component with at most $(2+\epsilon)k$ vertices will all correspond to one solution in the reduced instance.

Lemma 5.10. *For any constant $\epsilon > 0$, a reduced instance of* WDFAS*, formed by exhaustive application of Rules 1′, 2, and 3, has at most $(2+\epsilon)k$ vertices, where k is the minimum weight of a feedback arc set in the original instance.*

Proof. Each of the remaining strongly connected components has more than $\frac{2}{\epsilon}$ vertices. For V' the set of vertices in the reduced instance, at most $\lfloor \frac{|V'|}{\frac{2}{\epsilon}+1} \rfloor$ strongly connected components will remain. By Lemma 5.9, $|V'| \leq \lfloor 2k + \left(\frac{|V'|}{\frac{2}{\epsilon}+1} \right) \rfloor$, resulting in $|V'| \leq (2+\epsilon)k$, as required. □

Theorem 5.7. WDFAS *restricted to complete digraphs whose arc-weights satisfy probability and triangle inequality constraints admits a $(2+\epsilon)k$ vertex kernel for an arbitrarily small $\epsilon > 0$. The kernel can be computed in $O(n^2 + n \cdot \lfloor \frac{2}{\epsilon} \rfloor !)$ time. and the way we solve one does not affect the optimality of solutions in other components.*

Proof. Since Rules 1′, 2, and 3 are sound, we can always use them for reduction. According to Lemma 5.10, exhaustive application of Rules 1′, 2, and 3 reduces the number of vertices to at most $(2+\epsilon)k$ vertices.

Exhaustive application of Rule 1′ has the same effect as computing strongly connected components of $w_{<\frac{1}{2}}$, and thus can be implemented in $O(n^2)$ time. We can also keep track of the number of vertices in each strongly connected component, allowing us to identify small components within the same time bound. Application of Rule 3 for each component with at most $\frac{2}{\epsilon}$ vertices can be done in $O(\lfloor \frac{2}{\epsilon} \rfloor !)$ time if a brute-force search is used, yielding a total running time of $O(n^2 + n \cdot \lfloor \frac{2}{\epsilon} \rfloor !)$ in the worst case. □

Corollary 5.3. P-Kemeny Rank Aggregation *admits a partial kernel with at most $(2+\epsilon)\frac{k_t}{m}$ candidates for an arbitrarily small $\epsilon > 0$.*

5.4 Concluding Remarks

In the first section, we presented an $O^*(1.403^{k_t})$-time algorithm to find an optimal aggregation. Our algorithm improved the previous best running time of $O^*(1.53^{k_t})$ due to Betzler et al. [20]. The running time was subsequently decreased to $O^*(2^{O(\sqrt{\frac{k_t}{m}}\log\frac{k_t}{m})})$ [7, 68] and later to $O^*(2^{O(\sqrt{\frac{k_t}{m}})})$ [91].

We next developed an $O^*(4^{\frac{k_t}{m}})$-time algorithm to enumerate all optimal aggregations, giving the first FPT algorithm for Enum(P-Kemeny Rank Aggregation). In addition, we proved that there cannot be more than $4^{\frac{k_t}{m}}$ optimal aggregations for an instance of P-Kemeny Rank Aggregation and there are instances with this many optimal aggregations. We observed that our algorithm, as well as the analysis, worked for locally optimal aggregations.

In the final section, we obtained a $(2+\epsilon)\frac{k_t}{m}$-candidate partial kernel for P-Kemeny Rank Aggregation. Our kernelization is a direct implementation of the Extended Condorcet Criterion combined with a brute-force algorithm to solve small components (those including at most $\frac{2}{\epsilon}$ candidates) in the reduced instance. The brute-force algorithm is used in Theorem 5.7 to apply Rule 3. If this algorithm is replaced with the recent parameterized algorithm for WDFAS [91], a decreased ϵ (thus a decreased upper bound for the number of candidates) can be achieved within the same time bound. It is open whether the number of candidates can be decreased to $(2+o(1))\frac{k_t}{m}$.

Chapter 6

Parameterized Partial Enumerability

In this chapter, we establish the following connections between P-Neighbour String, P-Closest String, and P-Kemeny Rank Aggregation and their associated partial enumeration problems:

1. If an algorithm solves P-Neighbour String in time $t(|\mathcal{I}|, \max d_i)$ for any instance $\mathcal{I} = \{(s_1, d_1), \ldots, (s_n, d_n)\}$, $s_i \in \Sigma^\ell$ for all i, and for a non-decreasing function t, then there exists an algorithm that solves Partial(P-Neighbour String) in time $O(K \cdot |\Sigma| \cdot \ell \cdot t(|\mathcal{I}|, \max d_i))$.

2. If an algorithm solves P-Closest String in time $t(|\mathcal{I}|, d)$ for any instance $\mathcal{I} = \{(s_1, d), \ldots, (s_n, d)\}$, $s_i \in \Sigma^\ell$ for all i, and for a non-decreasing function t, then there exists an algorithm that solves Partial(P-Closest String) in time $O(K \cdot |\Sigma| \cdot \ell \cdot 2^d \cdot t(|\mathcal{I}|, d))$.

3. If an algorithm solves P-Kemeny Rank Aggregation in time $t(|\mathcal{I}|, \frac{k_t}{m})$ for any multi-set $\mathcal{I}$ of m total orders of $\tau(\sigma, \mathcal{I}) = k_t$, then there exists an algorithm that solves Partial(P-Kemeny Rank Aggregation) in time $O(K \cdot n^2 \cdot t(|\mathcal{I}|, \frac{k_t}{m}))$.

The time bounds are summarized in Table 6.1. Since P-Neighbour String is more general than P-Closest String, the extra 2^d factor in the time bound of Partial(P-Closest String) might seem strange. We have not tried to make this factor optimal. Nevertheless, the t functions in the two cases are not necessarily the same, and we expect to have a smaller time bound to produce one solution to P-Closest String.

These results are all based on a standard backtracking algorithm [121], which uses prefix problems to prune all branches with no solution.

Problem	Time for 1 solution	Time for K solutions
P-Neighbour String	$t(\lvert\mathcal{I}\rvert, \max_i d_i)$	$O(K \cdot \lvert\Sigma\rvert \cdot \ell \cdot t(\lvert\mathcal{I}\rvert, \max_i d_i))$
P-Closest String	$t(\lvert\mathcal{I}\rvert, d)$	$O(K \cdot \lvert\Sigma\rvert \cdot \ell \cdot 2^d \cdot t(\lvert\mathcal{I}\rvert, d))$
P-Kemeny Rank Aggregation	$t(\lvert\mathcal{I}\rvert, \frac{k_t}{m})$	$O(K \cdot n^2 \cdot t(\lvert\mathcal{I}\rvert, \frac{k_t}{m}))$

Figure 6.1: Time to solve Partial problems associated with P-Neighbour String, P-Closest String, and P-Kemeny Rank Aggregation

Inspired by this backtracking technique, we obtain a necessary and sufficient condition for instances of k-subset problems, a very common form of search problem, to have $f(k)$ solutions. The condition translates to the famous Erdös–Rado Theorem. We also demonstrate that the Erdös–Rado Conjecture (a stronger form of the Erdös–Rado Theorem) restricts $f(k)$ to c^k for certain k-subset problems.

Next, we show how a classical algorithm due to Lawler [101] can be used for an FPT enumeration of K best solutions for increasing k-subset problems, a common form of optimization problem. The time complexity almost matches the time complexity of finding a single solution for the particular increasing k-subset problem. As a result, we improve a few partial enumeration results. The results can be seen as the first step towards answering the question raised by Chen et al. [33], regarding the connection of FPT optimization problems (optimization problems whose corresponding search problems are in FPT) and optimization problems that have FPT partial enumerations.

We present the algorithm for partial enumeration in Section 6.1. The algorithm produces K solutions to a search problem R in FPT time assuming that it has access to FPT algorithms for both R and Prefix(R). The algorithm is extended to optimization problems in Section 6.3. We present parameterized reductions from Partial(P-Neighbour String), Partial(P-Closest String), and Partial(P-Kemeny Rank Aggregation) to P-Neighbour String, P-Closest String, and P-Kemeny Rank Aggregation in Section 6.1.1, based on which the FPT algorithm of Section 6.1 works for P-Neighbour String, P-Closest String, and P-Kemeny Rank Aggregation. In Section 6.2, we discuss a modification of the partial enumeration algorithm, and provide intuition concerning when the number of nodes in the corresponding execution tree, and thus the number of solutions, is bounded by $f(k)$, and when we expect this number to be bounded by c^k, for some constant c.

6.1 Prefix and Partial Problems

Algorithm 11 is a standard backtracking algorithm. It essentially produces every string p such that $(x, p) \in \textsc{Prefix}(R)$ and puts those in $R(x)$ in O, taking an early exit if the number of strings in O reaches K. The prefix checking at line 3 enables the algorithm to stop the search when there are no more strings in $R(x)$ that have prefix p. In this sense, the use of prefix problems is similar to their usage in showing the connection between the PC (Polynomial-time Check) and PF (Polynomial-time Find) complexity classes in Section 2.1.2, and the PC and P-enumerable classes in Section 2.1.4.

The returned strings are all distinct since different prefixes are passed to different nodes: prefixes at a node's descendants are always longer than the prefix at the node itself. Furthermore, since the prefixes passed to a node's children are always different, and none is a substring of another, strings with common prefixes all appear in the same branch. Therefore, the algorithm produces K disjoint strings in $R(x)$.

Algorithm 11: PartialEnum

```
  Require: A search problem $R$, $x$ and $p$ in $\Sigma^*$, and an integer $K$;      /* initially
           called with $p = \lambda$ (the empty string) */
1 $O \leftarrow \emptyset$;
2 if $K \leq 0$ then return $O$;
3 if $(x, p) \notin \textsc{Prefix}(R)$ then return $O$;
4 if $(x, p) \in R$ then insert $p$ in $O$;
5 foreach $p' \in \{p \cdot c : c \in \Sigma\}$ do
6     $O \leftarrow O \cup \text{PartialEnum}(R, x, p', K - |O|)$;
7 end
8 return $O$;
```

Observation 6.1. *If* $\textsc{Prefix}$*(R) can be checked in time* $t(|x| + |y|)$ *for a search problem* R *and for any input* (x, y)*, then* $\textsc{Partial}$*(R) can be solved in time* $O(K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y))$ *for any input* $(x, 1^K)$*, where* $Y = \max\{|y| : y \in R(x)\}$*.*

Proof. Using strong induction on the length of p, it is not hard to prove that PartialEnum(R, x, p, K) returns $K' \leq K$ solutions in $R(x)$ in time $O(K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y))$, where K' is the minimum of K and the number of solutions in $R(x)$ that have prefix p. Consequently, PartialEnum(R, x, λ, K) returns K solutions (or all solutions, if their number is smaller than K) in $R(x)$ within the required time.

Consider the execution tree of PartialEnum(R, x, λ, K). The number of solutions produced is at most K. The p values at ancestors of a node are all prefixes of the p value at the node. Therefore, every node in which a solution is found (line 4), has less than $Y = \max\{|y| : y \in R(x)\}$ ancestors. We call any such node a *solution node.* On the other hand, every internal node N is an ancestor of a solution node; otherwise, $(x, p) \notin \textsc{Prefix}(R)$ for the p value at the node N and the algorithm returns at line 3 without producing any children. Consequently, there are at most $K \cdot Y$ internal nodes: if we mark all ancestors of the at most K solution nodes, all the internal nodes will be marked, and therefore the internal nodes are among the at most $K \cdot Y$ nodes that are marked. Since each internal node has $|\Sigma|$ children, there are at most $K \cdot |\Sigma| \cdot Y$ nodes in the execution tree. The time spent in each node is bounded by $O(t(|x| + Y))$, making a total running time of $O(K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y))$, as required. □

To study partial enumeration of solutions for parameterized problems, we restate Observation 6.1 for 2-parameter time bounds:

Observation 6.2. *Suppose that R is a search problem parameterized by a parameter function κ. If $\textsc{Prefix}(R)$ can be checked in time $t(|x| + |y|, \kappa(x))$ for any input $((x, \kappa(x)), y)$, then $\textsc{Partial}(R)$ can be solved in time $O(K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y, \kappa(x)))$ for any input $((x, 1^K), \kappa(x))$, where $Y = \max\{|y| : y \in R(x)\}$.*

Corollary 6.1. *If $\textsc{Prefix}(R)$ is in FPT for a parameterized search problem R, then $\textsc{Partial}(R)$ is in FPT.*

In many cases the prefix problems of FPT problems are also in FPT. In fact, the input prefixes often prune the search space and make the search even more efficient.

6.1.1 Reducible Prefix Problems

For many problems, the assumption of having an efficient algorithm for prefix problems is not stronger than assuming an efficient algorithm for original problems. For example, finding a maximum-weight forest that includes certain edges and excludes certain edges can still be solved by Kruskal's algorithm (with slight modifications) [135]. Or, prefix problems of Fagin-definable problems [71] are not harder than themselves, since any assignment of a few variables to true or false will only simplify the formula.

Nevertheless, from the theoretical standpoint, we would like to show reductions from the prefix of a problem to the original problem.

Observation 6.3. *Suppose that R is a parameterized search problem in FPT. If* Prefix*(R) has a parameterized reduction to R, then* Partial*(R) is in FPT.*

In the following, we provide such reductions for Prefix(P-Neighbour String), Prefix(P-Closest String), and Prefix(P-Kemeny Rank Aggregation):

Lemma 6.1. Prefix*(*P-Neighbour String*) has a parameterized reduction to* P-Neighbour String*.*

Proof. Let $\mathcal{C} = ((\{(s_1, d_1), \ldots, (s_n, d_n)\}, \max_i d_i), y')$ be an instance of Prefix(P-Neighbour String). Let ℓ denote the length of the strings $s_1, s_2, \ldots s_n$ and ℓ' denote the length of y'. Let R be the region $\{\ell' + 1, \ell' + 2, \ldots, \ell\}$.

We can ignore the given prefix y' as long as we update the distances accordingly: $\mathcal{C}$ is a solution to Prefix(P-Neighbour String) if and only if $(\{(s'_1, d'_1), \ldots, (s'_n, d'_n)\}, \max_i d'_i)$ is a solution to Prefix(P-Neighbour String), where $s'_i = s_i|R$ and $d'_i = d_i - H(s_i|\overline{R}, y')$ for every i. □

Lemma 6.2. *There exists a parameterized reduction from* P-Neighbour String *to* P-Closest String*.*

Proof. Let $\mathcal{C} = (\{(s_1, d_1), \ldots, (s_n, d_n)\}, d)$ be an instance of P-Neighbour String, $d = \max_i d_i$. Also, let Δd_i denote $d - d_i$ and let Δd denote the maximum among the Δd_i's.

In order to reduce this instance to an instance of P-Closest String, we would like to make all the distances equal. We do so by making two copies of each string s_i, and adding prefixes at the beginning of these copies which force a distance increase of $d - d_i$ for at least one of the copies: $\mathcal{C}$ is a solution to P-Neighbour String if and only if $(\{(s'_1, d), \ldots, (s'_n, d), (s''_1, d), \ldots, (s''_n, d)\}, d)$ is a solution to P-Closest String, where $s'_i = (01)^{\Delta d_i}(00)^{\Delta d - \Delta d_i} s_i$ and $s''_i = (10)^{\Delta d_i}(00)^{\Delta d - \Delta d_i} s_i$. □

We use a combination of the reductions in Lemmas 6.1 and 6.2 to construct a reduction from Prefix(P-Closest String) to P-Closest String:

Lemma 6.3. Prefix*(*P-Closest String*) has a parameterized reduction to* P-Closest String*.*

Proof. An instance $((\mathcal{I}, d), y')$ is in Prefix(P-Closest String) if and only if it is in Prefix(P-Neighbour String). Therefore, any reduction from Prefix(P-Neighbour String) to Closest String will also be a reduction from Prefix(P-Closest String) to Closest String. Lemma 6.1 gives a parameterized reduction f_1 from Prefix(P-Neighbour String) to P-Neighbour String. Lemma 6.2 gives a parameterized reduction f_2 from P-Neighbour String to P-Closest String. The function $f_2 o f_1$ is thus a parameterized reduction from Prefix(P-Closest String) to P-Closest String. □

In our discussion for P-Kemeny Rank Aggregation, we assume that the input/output total orders (all in Total(U)) are represented by strings over the alphabet $\Sigma = U$. For example, the total order $\pi \in \text{Total}(\{a, b, c\})$ that orders a, b, and c as $b <_\pi a <_\pi c$ is represented as "bac".

Lemma 6.4. Prefix*(*P-Kemeny Rank Aggregation*) has a parameterized reduction to* P-Kemeny Rank Aggregation*.*

Proof. Let $\mathcal{C} = ((\mathcal{I}, \frac{k_t}{m}), y')$ be an instance of Prefix(P-Kemeny Rank Aggregation). Let U' be the subset of candidates contained in the prefix y'.

Having y' as a prefix forces all the candidates in U' be ordered before all other candidates. Therefore, we can remove the candidates in U' as long as we update the total distance k_t accordingly: $\mathcal{C}$ is a solution to Prefix(P-Kemeny Rank Aggregation) if and only if $(\mathcal{I}', \frac{k'_t}{m})$ is a solution to Prefix(P-Kemeny Rank Aggregation), where $\mathcal{I}'$ is constructed by removing the candidates in U' from $\mathcal{I}$ and k'_t is the adjusted distance $k_t - \sum_{a \in U', b \in U \setminus U'} |\mathcal{I}_{(b,a)}|$. □

6.2 A Note on Parameterized Enumerability

In the PartialEnum algorithm (Algorithm 11), the branching (i.e. ways of partitioning the space of solutions) could be changed in various ways. Similar branching techniques have been used by Murty [112], Lawler [101], Valiant [130], Carraresi and Sodini [29], and Hamacher [84]. In this section, we provide an alternative branching for k-subset problems, a common form of search problem. We characterize those problems that have $f(k)$ solutions, for some function f, based on this alternative branching.

Definition 6.1. *A* k-subset problem *is a search problem R such that for every input x every $y \in R(x)$ is a subset of size at most k.*

We use $\text{Domain}_R(x)$ to denote $\bigcup_{y \in R(x)} y$. For instance, when R is the k-subset problem mapping every graph x to the set of its k-vertex covers, $\text{Domain}_R(x)$ is the set of vertices in the graph x. We represent each solution $y \in R(x)$ as a string $y_1 y_2 \cdots y_{|\mathrm{Domain}_R(x)|}$ such that $y_i = 1$ if the ith element of the domain is in y and $y_i = 0$ otherwise.

Therefore, the alphabet Σ is restricted to $\{0, 1\}$ in all our discussions on k-subset problems.

We use $1|_s$ (respectively, $0|_s$) to denote the set of positions in which a string s has value 1 (respectively, 0), and use $\star$ to denote a character in the prefix that matches any character. Therefore, the solution represented by a string s is in fact $1|_s$. We also recall (from Chapter 2) that for a set of positions L and strings s_1 and s_2 of lengths $|L|$ and $\ell - |L|$, the notation $s_1 \oplus_L s_2$ is defined as the string s of length ℓ such that $s|L = s_1$ and $s|\overline{L} = s_2$.

6.2.1 The Connection to the Erdös–Rado Theorem

In this section, we show that the number of solutions for instances of k-subset problems is bounded by a function of k, if and only if a parameter Δ of the input instances is bounded by a function of k.

Algorithm AlternativeEnum, illustrated in Figure 12, is a modification of Algorithm 11 which produces all solutions (i.e. K is removed from the algorithm) and uses a special kind of branching for k-subset problems. The Find algorithm called is Algorithm 1 in Section 1, which returns a solution $y \in R(x)$ that matches the prefix p or returns $\perp$ when there is no such solution; this algorithm uses arbitrary algorithms to check $\textsc{Prefix}(R)$ and R. Algorithm AlternativeEnum uses an extended $\textsc{Prefix}(R)$ where prefixes are allowed to contain "don't care" $\star$'s. Note that we are mainly concerned with the number of solutions here, and do not really need these prefix algorithms or the algorithm to check R.

The idea is to branch based on the positions in $1|_s$ for a solution s. In particular, the algorithm sets aside the positions in $1|_p \cup 0|_p$, which are fixed already by the prefix p, from $1|_s$ (line 5). It then fixes the 1st, 2nd, . . . , or the last position in $1|_s \setminus (1|_p \cup 0|_p)$ to be 1 in the next solutions (by making it one in the new prefix) or to force the next solutions to be all zeros in $1|_s$ (line 6). Since s is a solution for a k-subset problem, $1|_s$ is ensured to have a cardinality at most k, thus a limited number of branches.

The new branching ensures two main properties in all the nodes in the execution tree of AlternativeEnum(x, λ), for any input x:

Lemma 6.5. *There are at most $k + 1$ branches from each node.*

Algorithm 12: AlternativeEnum

```
    Require: Two strings $x$ and $p$ in $\Sigma^*$;      /* initially called with $p = \lambda$ (the
             empty string) */
1   $O \leftarrow \emptyset$;
2   $y \leftarrow \text{Find}(x, p)$;
3   if $y = \perp$ then return $O$;
4   insert $y$ in $O$;
5   $L \leftarrow 1|_y \setminus (1|_p \cup 0|_p)$;
6   foreach $p' \in \{w \oplus_L p : |w| = |L|$ and $w \in (0^*1\star^*) \cup 0^*\}$ do      /* $w$ is a string
    over the alphabet $\{0, 1, \star\}$ */
7       $O \leftarrow O \cup \text{AlternativeEnum}(x, p')$;
8   end
9   return $O$;
```

Proof. The branches from a node correspond to the strings $w \in (0^*1\star^*) \cup 0^*$, of length $|L|$, produced at line 6. There are $|L|+1$ such w's. Since $y \in R(x)$ (guaranteed by the algorithm Find), and R is a k-subset problem, $|L|$ is always less than or equal to k. Therefore, there are $|L| + 1 \leq k + 1$ branches from each node, as required. □

As a consequence, the number of nodes in the execution tree is a function of k if and only if the depth of the tree depends only on k.

During the course of the algorithm, whenever a recursive call is made with $w = 0^{|L|}$, we say that the algorithm is taking a rightmost branch. For other recursive calls, we say that the algorithm is taking a non-rightmost branch.

Lemma 6.6. *In all but one branch (the rightmost branch), in each node, the number of* 1*'s in the prefix, i.e. p, is increased by one.*

Proof. By the definition of L, none of the positions in L are in $1|_p$. Since in every non-rightmost branch, w has a 1, the number of 1's in $w \oplus_L p$ is one plus the number of 1's in p. □

Since the algorithm looks for subsets of size at most k (represented by binary strings with at most k 1's), Lemma 6.6 ensures that the number of times the algorithm goes through non-rightmost branches is limited to k.

On the other hand, we show that the number of consecutive rightmost branches taken is bounded by $\Delta(R(x))$, where a Δ-*system* [61] (also known as a *sunflower*) is a family of sets all of which share a subset Y and none of their pairwise intersections includes an element outside Y, and $\Delta(\mathcal{S})$ for a family of sets $\mathcal{S}$ denotes the cardinality of a largest Δ-system that is a subset of $\mathcal{S}$:

Lemma 6.7. *Suppose that there are t consecutive rightmost branches taken in the execution tree of AlternativeEnum(x, λ), for some integer t. Then, $R(x)$ includes a Δ-system of cardinality $t + 1$.*

Proof. Consider the node N_0 at which these t consecutive rightmost branches begin. Let N_i, $1 \le i \le t$, be the node executed after taking the rightmost branch in N_{i-1}, and let y_i, $0 \le i \le t$, denote the y computed at line 2 in N_i. We show that $S = \{1|_{y_i} : 0 \le i \le t\}$ is a sunflower contained in $R(x)$.

It is guaranteed by the algorithm Find that $1|_y \in R(x)$ for every y computed at line 2. Therefore, it suffices to show that S is a sunflower. Assume that p_i is the value of p passed to the node N_i. Due to the removal of $1|_p$ and $0|_p$ in the construction of L (line 5), the algorithm does not change any 0's or 1's fixed in a prefix, thus $1|_{p_t} \subseteq 1|_{p_{t-1}} \subseteq \cdots \subseteq 1|_{p_0}$ and $0|_{p_0} \subseteq 0|_{p_1} \subseteq \cdots \subseteq 0|_{p_t}$. On the other hand, in a rightmost branch $w = 0^{|L|}$, and therefore, no 1 is added to the current prefix, proving that $1|_{p_0} = 1|_{p_1} = \cdots = 1|_{p_t}$. We claim that $1|_{y_i}$'s do not intersect in a position outside $1|_{p_0}$. Assume instead that $1|_{y_j} \setminus 1|_{p_0}$ and $1|_{y_i} \setminus 1|_{p_0}$ share a position $u \notin 1|_{p_0}$, for some $1 \le j < i \le t$. By the definition, u will be in the L constructed in N_j, since $L = 1|_{y_j} \setminus (1|_{p_j} \cup 0|_{p_j}) = 1|_{y_j} \setminus (1|_{p_0} \cup 0|_{p_j})$. We know that $p_{j+1} = 0^{|L|} \oplus_L p_j$. Therefore, the value of p_{j+1} in u will be 0, and thus $u \in 0|_{p_{j+1}} \subseteq 0|_{p_i}$. This is in contradiction with $u \in 1|_{p_i}$ in the definition of u. The resulting contradiction proves that S forms a sunflower. □

As a consequence of Lemmas 6.6 and 6.7, the depth of the execution tree will be bounded by $(k + 1)\Delta(R(x))$:

Theorem 6.1. *Suppose that R is a k-subset problem. Then, $|R(x)| \le f(k)$ for some function f if and only if $\Delta(R(x)) \le g(k)$ for some function g.*

Proof. If $\Delta(R(x))$ exceeds any function of k, it means that $R(x)$ contains a sunflower S whose cardinality exceeds any function of k. Therefore, $|R(x)|$ exceeds any function of k.

On the other hand, if $\Delta(R(x)) \le g(k)$ for some function g, consider a deepest leaf in the execution tree of AlternativeEnum(x, λ). Suppose that the leaf is reached from the root of the tree by taking a sequence of non-rightmost or rightmost branches represented by $d_1 d_2 \cdots d_a$, where $d_i \in \{n, r\}$. As a corollary of Lemma 6.6, there are at most k n's in this sequence, since each n increases the number of 1's in the prefix and the algorithm will return at line 3 whenever p contains $k+1$ 1's. In addition, by Lemma 6.7, there cannot be $\Delta(R(x))$ consecutive r's in this sequence. Therefore, the length of the sequence of d's, thus the depth of the execution tree, is at most $(k+1)\cdot\Delta(R(x)) \leq (k+1)\cdot g(k)$. Combined with Lemma 6.5, this proves that the number of nodes in the execution tree, i.e. $|R(x)|$, is a function of k. □

Theorem 6.1 is indeed a corollary of the Erdös–Rado Theorem, essentially saying that set systems of large cardinality include large Δ-systems. Let $r(\mathcal{S})$ denote the cardinality of a largest set in a family $\mathcal{S}$ of sets. Then, the Erdös–Rado Theorem states that

Theorem 6.2. *[61] Suppose that $\mathcal{S}$ is a family of sets such that $r(\mathcal{S}) \leq k$ and $\Delta(\mathcal{S}) \leq t$. Then, the cardinality of $\mathcal{S}$ is at most $k!t^k(1-\frac{1}{2!t}-\frac{2}{3!t^2}-\cdots-\frac{k-1}{k!t^{k-1}})$.*

If $f(k,t)$ denotes the maximum cardinality of the set systems in $\{\mathcal{S} : r(\mathcal{S}) \leq k, \Delta(\mathcal{S}) \leq t\}$, Theorem 6.2 can also be stated as

$$f(k,t) \leq k!t^k\big(1-\frac{1}{2!t}-\frac{2}{3!t^2}-\cdots-\frac{k-1}{k!t^{k-1}}\big).$$

Notice that the proof of Theorem 6.1 is also giving an upper bound for $f(k,t)$. Nevertheless, the bound is not as precise as the upper bound in the Erdös–Rado Theorem.

6.2.2 Examples

In the previous section, we showed how $\Delta(R(x))$ affects the number of solutions in $R(x)$. In the following, we show that $\Delta(R(x))$ is a natural parameter in k-subset problems. In particular, we show that $\Delta(R(x))$ is a constant when $R(x)$ is the set of all minimal k-vertex covers, k-feedback arc sets on tournaments, or all minimal solutions to k-Cluster Deletion (minimal sets of edges whose removal from the input graph transforms it to a union of cliques).

Theorem 6.3. *Suppose that $R(x)$ is the set of all minimal k-vertex covers in the undirected graph x. Then, $\Delta(R(x)) \leq 2$.*

Proof. Assume instead that $R(x)$ includes a Δ-system $\mathcal{S} = \{\sigma_1, \sigma_2, \sigma_3\}$ all intersecting in a set Y. Since the three sets in $\mathcal{S}$ are distinct, $\sigma_i \setminus Y \neq \emptyset$ for some $1 \leq i \leq 3$. Due to the minimality of σ_i, Y cannot be a feasible solution. Hence, we can assume that there is an edge e in the input graph not covered by Y. To cover e, $\sigma_i \setminus Y$ must include one of its endpoints. Similarly, the two other vertex covers in $\mathcal{S}$ must also include an endpoint of e. However, $(\sigma_1 \setminus Y)$, $(\sigma_2 \setminus Y)$, and $(\sigma_3 \setminus Y)$ are disjoint, which is not possible as there are only two endpoints of e to be assigned to three sets. The resulting contradiction proves that $\Delta(R(x)) \leq 2$. □

Theorem 6.4. *Suppose that $R(x)$ is the set of all minimal k-feedback arc sets in the tournament graph x. Then, $\Delta(R(x)) \leq 3$.*

Proof. Assume instead that $R(x)$ includes a Δ-system $\mathcal{S} = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ all intersecting in a set Y. Again, since the four sets in $\mathcal{S}$ are distinct, Y cannot be a feasible solution. Hence, we can assume that reversing the arcs in Y does not make x acyclic. Since the resulting graph is still a tournament, it has to have a three-cycle C. To remove this cycle, $\sigma_i \setminus Y$ must include one of the arcs in C, for every $1 \leq i \leq 4$. However, $(\sigma_1 \setminus Y)$, $(\sigma_2 \setminus Y)$, $(\sigma_3 \setminus Y)$, and $(\sigma_4 \setminus Y)$ are disjoint, which is not possible as there are only three arcs in C. The resulting contradiction proves that $\Delta(R(x)) \leq 3$. □

Theorem 6.5. *Suppose that $R(x)$ is the set of all minimal solutions to an input instance x of k-Cluster Deletion. Then, $\Delta(R(x)) \leq 2$.*

Proof. Assume instead that $R(x)$ includes a Δ-system $\mathcal{S} = \{\sigma_1, \sigma_2, \sigma_3\}$ all intersecting in a set Y. Since the three sets in $\mathcal{S}$ are distinct, Y cannot be a feasible solution. Hence, we can assume that removing the edges in Y in the input graph x results in a graph x' that is not a union of a set of cliques. As a consequence, x' has an induced path of length 2. To convert x' into a union of cliques, $\sigma_i \setminus Y$ must include one of the edges in this path, for every $1 \leq i \leq 3$. However, $(\sigma_1 \setminus Y)$, $(\sigma_2 \setminus Y)$, and $(\sigma_3 \setminus Y)$ are disjoint, which is not possible as there are only two edges in the path. The resulting contradiction proves that $\Delta(R(x)) \leq 2$. □

6.2.3 Implications of the Erdös-Rado Conjecture

Erdös and Rado conjectured that $f(k,t)$ has an exponential bound [61], when t is a constant:

Conjecture 6.1. *[61] For every $t \geq 1$, there exists a constant c_t such that for every $k \geq 1$*

$$f(k,t) \leq (c_t \cdot t)^k.$$

The conjecture has been open for the past 50 years. There has been no success in proving the bound even for $t = 2$, i.e., the smallest t producing non-trivial $f(k,t)$'s. In fact, despite being an old result, improvements made to the bounds for $f(k,t)$ have only been on the coefficient of t^k for certain values of k [1, 3, 2, 97, 96, 98]

Nevertheless, if the conjecture is true, it will give upper bounds on the number of solutions for k-subset problems of bounded largest sunflower.

Theorem 6.6. *Suppose that R is a k-subset problem. Then, $|R(x)| \leq (c \cdot \Delta(R(x)))^k$, for some constant c, if the Erdös–Rado Conjecture holds.*

None of the examples of Section 6.2.2 showcase the effectiveness of the Erdös-Rado conjecture in obtaining an exponential bound. In all these constant-bounded $\Delta(R(x))$ examples, the bound for $\Delta(R(x))$ is proved based on a constant-sized structure with which any $y \in R(x)$ has to share an element. Therefore, the structure gives an immediate branching algorithm (with constant branching factor), and an immediate exponential upper bound for $|R(x)|$, with no need to use the conjecture.

Nonetheless, we expect the conjecture to give new insight for some problems. Roughly speaking, since the Erdös-Rado conjecture is a long-time conjecture, we expect the c^k bound be hard to derive for some families of sets of bounded largest sunflower.

6.3 Partial Problems of Optimization Problems

A classical algorithm of Lawler solves Partial problems of certain optimization problems [101]. Algorithm PartialBestEnum (Algorithm 14) is a modification of PartialEnum (Algorithm 11 in Section 6.1) according to Lawler's algorithm: in this algorithm, the branching is not always made from the most recent node produced. The prefix used to construct the branches (Algorithm 13, line 4) is chosen based on how good the best solution in each branch is. Consequently, instead of a prefix check, the algorithm needs to find a best solution that matches each prefix. The best solutions are then compared (Algorithm 13, line 4), the best solution among them is chosen as the next best solution (Algorithm 13, line 6), and its corresponding prefix is used for branching (Algorithm 13, line 7).

Although Algorithm PartialBestEnum does not rely on a parameter, we state the theorems in the parameterized form:

Lemma 6.8. *If* Search*(*Prefix*(Q)), for an optimization problem Q, can be solved in time $t(|x| + |y|, \kappa(x))$ for a function t and for any input (x, y) and a parameter function κ, then* Partial*(Q) is solvable in time $O(K \log K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y, \kappa(x)))$ for any input $(x, 1^K)$, where $Y = \max\{|y| : y \in S(x)\}$.*

Proof. We associate a node in the execution tree of PartialBestRec with the (p, y) chosen at line 4. A solution y associated with a node is always a best solution that matches its associated prefix p, thus a best solution among solutions associated with potential descendants of the node. As a result, the solutions stored in P are the best solutions that might be found in future branchings. Therefore, the algorithm selects the best solution in P as the next best solution (lines 4 and 6).

A solution cannot be associated with more than Y nodes, as it matches at most Y prefixes. Therefore, a new solution will be put in O in every Y rounds, resulting in a total of at most KY internal nodes and at most $KY \cdot |\Sigma|$ nodes in total. If the (p, y) pairs in P are stored in a heap, line 4 is computed in $O(\log K)$ time, making an overall running time of $O(K \log K \cdot |\Sigma| \cdot Y \cdot t(|x| + Y, \kappa(x)))$, as required. □

Corollary 6.2. *If* Search*(*Prefix*(Q)), for an optimization problem Q, is in FPT when parameterized by a parameter function κ, then* Partial*(Q) is in FPT when parameterized by $\kappa' : (x, 1^K) \mapsto \kappa(x)$.*

Algorithm 13: PartialBestRec

```
   Require: A string $x$, and a set $P$ of string pairs, an integer $K$, and a measure
            function $m$
1  $O \leftarrow \emptyset$;
2  if $K \leq 0$ then return $O$;
3  if $P = \emptyset$ then return $O$;
4  choose $(p, y) \in P$ that minimizes $m(x, y)$;
5  remove $(p, y)$ from $P$;
6  insert $y$ into $O$;
7  foreach $p' \in \{p \cdot c : c \in \Sigma\}$ do
8      find $y'$ such that $((x, p'), y') \in$ Search(Prefix($Q$));
9      if $y' \neq y$ was found then insert $(p', y')$ in $P$;
10 end
11 return $O \cup$ PartialBestRec($x, P, K - 1, m$);
```

Algorithm 14: PartialBestEnum

```
Require: A string $x$, an integer $K$, and a measure function $m$
1 $P \leftarrow \emptyset$;
2 find $y$ such that $((x, \lambda), y) \in$ SEARCH(PREFIX($Q$));
3 if $y \neq \perp$ then insert $(\lambda, y)$ in $P$;
4 return PartialBestRec($x, P, K, m$);
```

6.3.1 Increasing k-Subset Problems

The significance of Corollary 6.2 becomes clearer when we notice that PARTIAL(Q) has the same time complexity as Q for a large group of optimization problems:

Definition 6.2. *An* increasing k-subset problem *is an optimization problem $\mathcal{Q} = (\mathcal{I}, S, m)$ for which there exists a domain set D and a weight function $w : D \mapsto \mathbb{R}$ such that every $y \in S(x)$ is a subset of size at most k in D and $m(x, y) = q_x(\sum_{e \in y} w(e))$ for an increasing function q_x.*

We represent each solution $y \in S(x)$ as a string $y_1 y_2 \cdots y_{|D|}$ such that $y_i = 1$ if the ith element of the domain is in y and $y_i = 0$ otherwise.

The definition matches definitions of WEIGHTED k-VERTEX COVER, WEIGHTED k-FVS, WEIGHTED k-DOMINATING SET, and k-EDGE DOMINATING SET problems [135]. Considering a negated weight function, the definition also matches that of WEIGHTED k-PATH, where the goal is to find a k-path of largest total arc-weights.

Lemma 6.9. *If* SEARCH*(Q) can be solved in time $t(|x|, k)$ for an increasing k-subset problem Q and for any input instance (x, k) of Q, then* SEARCH*(*PREFIX*(Q)) can be solved in time $O(t(|x|, k))$ for any input instance $((x, p), k)$.*

Proof. Consider an instance $((x, p), k)$ of SEARCH(PREFIX(Q)).

Suppose that w is the weight function associated with Q. The running time $t(|x|, k)$ is independent of weight values. Therefore, we can freely change w and expect the same time bounds.

As a consequence, SEARCH(Q') can be solved in time $t(|x|, k)$ for the new k-subset problem $Q' = (\mathcal{I}, S, m')$ and for any instance (x, k) of Q, where $m'(x, y) = q_x(\sum_{e \in y} w'(e))$ and the adjusted weight function $w' : D \mapsto \mathbb{R}$ is defined as $w'(e) = w(e)$ if $e \in 1|_p$, $w'(e) = w(e) + 2\epsilon$ if $e \in 0|_p$, and $w'(e) = w(e) + \epsilon$ otherwise, for an arbitrary constant $\epsilon > 0$.

```
Algorithm 15: ALTERNATIVEPARTIALBESTREC
7.1  $L \leftarrow 1|_y \setminus (1|_p \cup 0|_p)$;
7.2  foreach $p' \in \{w \oplus_L p : |w| = |L| \text{ and } w \in (0^+1\star^*) \cup 0^*\}$ do
```

We show that the instance $((x,p),k)$ of SEARCH(PREFIX(Q)) has a solution y if and only if y is a solution for (x,k) in SEARCH(Q'), if ϵ is chosen small enough. Suppose that ϵ is smaller than $m((x,k),y') - m((x,k),y)$ for a next best solution y' in Q. That is, $m((x,k),y') \neq m((x,k),y)$ and there is no solution $y'' \in S((x,k))$ that satisfies $m((x,k),y) < m((x,k),y'') < m((x,k),y')$. Then, it is not hard to see that the set of optimal solutions for (x,k) in Q' is the subset of optimal solutions for (x,k) in Q that begin with the prefix p. Therefore, if y is a solution for $((x,p),k)$ in SEARCH(PREFIX(Q)), it will be an optimal solution for (x,k) in Q', and thus a solution for (x,k) in SEARCH(Q'). Similarly, if y is a solution for (x,k) in SEARCH(Q'), it is an optimal solution for (x,k) in Q', and thus a solution for $((x,p),k)$ in SEARCH(PREFIX(Q)). □

Corollary 6.3. *Suppose that Q is an increasing k-subset problem. If* SEARCH*(Q) is in FPT when parameterized by k, then* SEARCH*(*PARTIAL*(Q)) is in FPT with respect to k.*

The running time of Lemma 6.8 can be slightly improved if the prefixes are allowed to contain a special $\star$ character that matches all characters:

Theorem 6.7. *If* SEARCH*(*PREFIX*(Q)) can be solved in time $t(|x|,k)$ for an increasing k-subset problem Q and any input instance (x,k) of Q and any prefix string over $\{0,1,\star\}$, then* PARTIAL*(Q) is solvable in time $O(K \log K \cdot k \cdot t(|x|,k))$ for any input $(x,1^K)$.*

Proof. It suffices to substitute line 7 in the PartialBestRec algorithm with the two lines in Algorithm AlternativePartialBestRec (Algorithm 15). □

The following partial enumeration algorithms are examples of Theorem 6.7's new results:

Weighted k-Edge Dominating Set. A partial enumeration algorithm due to Wang et al. [134] finds the K best k-edge dominating sets in a weighted graph in time $O(5.6^{2k}k^4K^2 + 4^{2k}nk^3K)$. If K is set to 1, this algorithm finds a minimum-weight k-edge dominating set in time $O(5.6^{2k}k^4 + 4^{2k}nk^3)$, based on which Theorem 6.7 gives an $O((5.6^{2k}k^4 + 4^{2k}nk^3) \cdot K \log K \cdot k)$-time partial enumeration algorithm, improving the time bound of Wang et al. [134].

Maximum-Weight k-Path. Using an $O(4^{k+O(\log^3 k)}nm)$-time algorithm of Chen et al. for finding a maximum-weight k-path [34], Theorem 6.7 computes the K heaviest k-paths in time $O(4^{k+O(\log^3 k)}K \log K \cdot knm)$. This is an improvement over the partial enumeration algorithm of Chen et al. [33] that runs in time $O(12.8^k + 6.4^k k^2 n^3 K)$ (for reasonable values of K).

Theorem 6.7 does not trivially extend to parameterizations of increasing subset problems by total weights of the solutions. An example is the Real-Weighted k-Vertex Cover [114] problem, which asks for a minimum weight vertex cover whose total weight of vertices does not exceed k, on the condition that the weights are real numbers greater than or equal to 1. The problematic issue is that the parameter might vary for the 1st, 2nd, and Kth solutions, thus making it hard to talk about the running time in terms of k (i.e., the first parameter).

6.4 Concluding Remarks

In this chapter, we presented FPT algorithms for Partial(P-Neighbour String), Partial(P-Closest String), and Partial(P-Kemeny Rank Aggregation), all based on standard backtracking techniques. As a matter of fact, we provided sufficient properties for search problems to have "efficient" partial enumeration algorithms. We then applied Lawler's idea [101] (mentioned in Section 6.3) to the backtracking algorithm to compute K best solutions in any increasing k-subset problem, in about the same time as finding a single solution for the problem. As a result, we improve the previous best time bounds to solve Partial(Maximum-Weight k-Path) and Partial(Weighted k-Edge Dominating Set).

We also observed that the Erdös–Rado Theorem categorizes k-subset problems: a k-subset problem has $f(k)$ solutions for any input x, for some function f, if and only if $\Delta(R(x)) \leq g(k)$ for some function g. Furthermore, a constant $\Delta(R(x))$, for every input x, implies a c^k number of solutions for x, for some constant c, if the Erdös–Rado Conjecture holds.

Chapter 7

Conclusions and Future Work

In the thesis, we developed the first FPT enumeration algorithms for P-Neighbour String and P-Kemeny Rank Aggregation in Chapters 4 and 5. Both our algorithms use polynomial space. When the input strings are all binary, our algorithm for P-Neighbour String can find a closest string in $O^*(5^{(1+\lambda)d})$ time, for arbitrary $\lambda > 0$, and thus improves (the asymptotic dependence on d of) the previous best time bound of $O^*(6.73^d)$ [38] for finding a single solution for Closest String. A future direction is to reduce the dependence on the size of the alphabet, such that the improvement extends to arbitrary alphabets.

Furthermore, we showed in Chapter 4 how a slight modification to the StringSearch algorithm of Ma and Sun [105] (for solving Closest String and Neighbour String) makes it enumerative, and achieved an $O^*(6^{d(1+\epsilon)})$, for arbitrary $\epsilon > 0$, time bound through a new analysis.

The previous time bound for StringSearch [105] also carries over to the modified algorithm. It is not hard to see that subsequent algorithms of Wang and Zhu [136], Zhao and Zhang [141], Chen and Wang [39], and Chen et al. [38] can also be made enumerative, with no change in their time bounds. As a consequence, new non-enumerating approaches are required in order to find $o^*(4^d)$-time algorithms for Closest String even for binary strings, since there are instances that have as many as 4^d closest strings (such as the instance including two binary strings 0^{2d} and 1^{2d}).

We also observed in Chapter 5 that our proposed $O^*(4^{\frac{k_t}{m}})$-time enumeration algorithm for P-Kemeny Rank Aggregation enumerates all locally optimal aggregations (a superset of the required set of all optimal aggregations). We emphasize that an $f(\frac{k_t}{m})n^{O(1)}$ upper bound on the number of locally optimal aggregations is surprising. A future direction can be looking for a new parameter that is more tuned to the complexity of enumerating all optimal aggregations rather than locally optimal aggregations.

The main contribution in the enumeration algorithm of Chapter 5 was the tight upper bound $4^{\frac{k_t}{m}}$ on the number of locally optimal aggregations. The algorithm itself could be replaced with a partial enumeration algorithm: there exists an $O(mn \log n)$-time algorithm [57] that computes a locally optimal aggregation, which can be adjusted to solve the corresponding prefix problem. Taking the approach of Chapter 6 (Algorithm 11) one can obtain an algorithm that computes K locally optimal aggregations in time $O(Kn^2(mn \log n))$. Therefore, to enumerate all the locally optimal aggregations, it suffices to set K to $4^{\frac{k_t}{m}}$, which results in a time bound of $O(4^{\frac{k_t}{m}} n^2(mn \log n)) \in O^*(4^{\frac{k_t}{m}})$. The same approach gives partial enumeration algorithms for P-Neighbour String, P-Closest String, and P-Kemeny Rank Aggregation. Combined with a classical algorithm due to Lawler [101], this also gives a partial enumeration algorithm for the family of increasing k-subset problems. Although the approach is not new, we are the first to use it as a general approach in the parameterized setting. The final theorem improves previous partial enumerations for Maximum-Weight k-Path and Weighted k-Edge Dominating Set.

We think the partial enumeration for P-Kemeny Rank Aggregation (obtained in Chapter 6) is of independent interest. As a result, the existing $O(2^{O(\sqrt{\frac{k_t}{m}})} + n^{O(1)})$-time algorithm for P-Kemeny Rank Aggregation [91] yields a subexponential-time algorithm to find a subexponential number of optimal aggregations. This is particularly useful since there might be exponentially many optimal aggregations (shown in Chapter 5), and thus a full enumeration is much more time-consuming.

In a slightly different direction, we obtained small partial kernels for P-Kemeny Rank Aggregation, improving the previous best partial kernel for the problem [18]. One of the two kernels we provide is enumerative, and thus can be combined with every enumeration algorithm for P-Kemeny Rank Aggregation. Possible extensions include empirical studies to compare the sizes of our partial kernels and the previous partial kernel of Betzler et al. [18]. In fact, the new sizes are always smaller, but an empirical study will show how much smaller they are on sample instances.

As a side note, we observed a connection between the famous theorem of Erdös and Rado [61] and the number of solutions in k-subset problems, a very common form of search problems including k-Vertex Cover, k-Feedback Vertex/Arc Set, and Cluster Deletion. As a consequence, the cardinality t of the largest sunflower contained in sets of solutions of input instances is a good indication of whether the number of solutions is bounded by some exponential function of k, bounded by some function of k, or not bounded by a function of k at all. However, since the upper bounds depend only on t and k, they are not using many properties of the problems. Examples show that $\Omega(t^k)$ is the best upper bound one can hope to get from this technique [61, 41]. Nonetheless, evaluating t values can be a first step to estimate the number of solutions for any k-subset problem.

In summary, our investigations on (partial) enumeration algorithms and enumerative kernels not only provided effective methods for finding multiple optimal solutions for P-Closest String, P-Neighbour String, and P-Kemeny Rank Aggregation, but also provided new insight into properties of optimal solutions. Our improved algorithm for Closest String for binary strings and our reduced $(2+\epsilon)\frac{k_t}{m}$-candidate partial kernel for P-Kemeny Rank Aggregation resulted from this new insight. It would be interesting to have empirical studies on this partial kernelization and the enumeration algorithms. Another natural direction to pursue is to investigate the complexity of enumerating representations of optimal solutions for these two problems. Studies on enumeration aspects of other parameterized problems are also directions for future research.

References

[1] H. L. Abbott and B. Gardner. On a combinatorial theorem of Erdös and Rado. *Recent progress in Combinatorics*, pages 211–215, 1969.

[2] H. L. Abbott and D. Hanson. On finite Δ-systems. *Discrete Math.*, 8:1–12, 1974.

[3] H. L. Abbott, D. Hanson, and N. Sauer. Intersection theorems for systems of sets. *Combin. Theory Ser. A*, 12:381–389, 1972.

[4] N. Ailon. Aggregation of partial rankings, p-ratings and top-m lists. *Algorithmica*, 57:284–300, 2010.

[5] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):1–27, 2008.

[6] J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33:461–493, 2002.

[7] N. Alon, D. Lokshtanov, and S. Saurabh. Fast FAST. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *ICALP '09: Proceedings of the 36th International Colloquium on Automata, Languages and Programming*, volume 5555, pages 49–58. Springer Berlin Heidelberg, 2009.

[8] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

[9] A. Andoni, P. Indyk, and M. Patrascu. On the optimality of the dimensionality reduction method. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 449–458, Washington, 2006. IEEE Computer Society.

[10] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[11] V. Arvind and V. Raman. Approximation algorithms for some parameterized counting problems. In Prosenjit Bose and Pat Morin, editors, *Algorithms and Computation*, volume 2518 of *ISAAC '02: Proceedings of the 13th International Symposium on Algorithms and Computation*, pages 169–189. Springer Berlin / Heidelberg, 2002.

[12] J. J. Bartholdi and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8:341–354, 1991.

[13] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[14] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[15] A. Ben-Dor, G. Lancia, J. Perone, and R. Ravi. Banishing bias from consensus sequences. In A. Apostolico and J. Hein, editors, *CPM '97: Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, volume 1264 of *LNCS*, pages 247–261, Heidelberg, 1997. Springer.

[16] S. Bessy, F. V. Fomin, S. Gaspers, C. Paul, A. Perez, S. Saurabh, and S. Thomassé. Kernels for feedback arc set in tournaments. *J. Comput. Syst. Sci.*, 77(6):1071–1078, 2011.

[17] N. Betzler, R. Bredereck, J. Chen, and R. Niedermeier. Studies in computational aspects of voting. In H. Bodlaender, R. Downey, F. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 318–363. Springer-Verlag, Berlin, Heidelberg, 2012.

[18] N. Betzler, R. Bredereck, and R. Niedermeier. Partial kernelization for rank aggregation: Theory and experiments. In V. Raman and S. Saurabh, editors, *IPEC '10: Proceedings of the 5th International Symposium on Parameterized and Exact Computation*, volume 6478 of *LNCS*, pages 26–37, 2010.

[19] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny scores. In *AAIM '08: Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 60–71, 2008.

[20] N. Betzler, M. R. Fellows, J. Guo, R. Niedermeier, and F. A. Rosamond. Fixed-parameter algorithms for Kemeny rankings. *Theor. Comput. Sci.*, 410(45):4554–4570, 2009.

[21] N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average parameterization and partial kernelization for computing medians. *J. Comput. Syst. Sci.*, 77(4):774–789, July 2011.

[22] T. C. Biedl, F. Brandenburg, and X. Deng. On the complexity of crossings in permutations. *Discrete Mathematics*, 309(7):1813–1823, 2009.

[23] D. Binkele-Raible and H. Fernau. Enumerate and measure: Improving parameter budget management. In *IPEC '10: Proceedings of the 3rd International Symposium on Parameterized and Exact Computation*, pages 38–49, 2010.

[24] J. Borda. *Mémoire sur les élections au scrutin.* Histoire de l'Académie Royale des Sciences, 1781.

[25] C. Boucher, D. G. Brown, and S. Durocher. On the structure of small motif recognition instances. In A. Amir, A. Turpin, and A. Moffat, editors, *SPIRE '08: Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, volume 5280 of *LNCS*, pages 269–281, Heidelberg, 2008. Springer.

[26] C. Boucher and M. Omar. On the hardness of counting and sampling center strings. In E. Chavez and S. Lonardi, editors, *SPIRE '10: Proceedings of the 17th International Symposium on String Processing and Information Retrieval*, volume 6393 of *LNCS*, pages 127–134, Heidelberg, 2010. Springer.

[27] F. Brandenburg, A. Gleißner, and A. Hofmeier. Comparing and aggregating partial orders with kendall tau distances. *Discrete Math., Alg. and Appl.*, 5(2), 2013.

[28] L. Cai and D. Juedes. On the existence of subexponential parameterized algorithms. *Journal of Computer and System Sciences*, 67:789–807, 2003.

[29] P. Carraresi and C. Sodini. An efficient algorithm for the bipartite matching problem. *Eur. J. Oper. Res.*, 23:86–93, 1986.

[30] I. Charon and O. Hudry. An updated survey on the linear ordering problem forweighted or unweighted tournaments. *Annals of Operations Research*, 175(1):107–158, 2010.

[31] C. R. Chegireddy and H. W. Hamacher. Algorithms for finding k-best perfect matchings. *Discrete Applied Mathematics*, 18(2):155–165, 1987.

[32] J. Chen, F. V. Fomin, Y. Liu, S. Lu, and Y. Villanger. Improved algorithms for feedback vertex set problems. *J. Comput. Syst. Sci.*, 74(7):1188–1198, 2008.

[33] J. Chen, I. A. Kanj, J. Meng, G. Xia, and F. Zhang. On the effective enumerability of NP problems. In *IWPEC '06: Proceedings of the 2nd International Workshop on Parameterized and Exact Computation*, volume 4169, pages 215–226, 2006.

[34] J. Chen, J. Kneis, S. Lu, D. Mölle, S. Richter, P. Rossmanith, S. Sze, and F. Zhang. Randomized divide-and-conquer: Improved path, matching, and packing algorithms. *SIAM Journal on Computing*, 38(6):2526–2547, 2009.

[35] J. Chen, Y. Liu, and S. Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, May 2009.

[36] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5), 2008.

[37] J. Chen and J. Meng. On parameterized intractability: Hardness and completeness. *The Computer Journal*, 51(1):39–59, 2008.

[38] Z. Chen, B. Ma, and L. Wang. A three-string approach to the closest string problem. *J. Comput. Syst. Sci.*, 78(1):164–178, 2012.

[39] Z. Chen and L. Wang. Fast exact algorithms for the closest string and substring problems with application to the planted (l, d)-motif model. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8:1400–1410, 2011.

[40] R. Chitnis, M. Hajiaghayi, and D. Marx. Fixed-parameter tractability of directed multiway cut parameterized by the size of the cutset. In *SODA '12: Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1713–1725. SIAM, 2012.

[41] V. Chvátal. On finite Δ-systems of Erdös and Rado. *Acta Mathematica Hungarica*, 21:341–355, 1970.

[42] M. Condorcet. *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix.* L'imprimerie royale, 1785.

[43] V. Conitzer, A. Davenport, and J. Kalagnanam. Improved bounds for computing Kemeny rankings. In *AAAI '06: Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, pages 620–626. AAAI Press, 2006.

[44] D. Coppersmith, L. K. Fleischer, and A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *SODA '06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 776–782, 2006.

[45] B. Courcelle. Graph rewriting: An algebraic and logic approach. *Handbook of Theoretical Computer Science*, pages 194–242, 1990.

[46] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1):23–52, 2001.

[47] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1–3):315–323, 2004.

[48] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theor. Comput. Sci.*, 351(3):337–350, 2006.

[49] P. Damaschke and L. Molokova. The union of minimal hitting sets: Parameterized combinatorial bounds and counting. *Journal of Discrete Algorithms*, 7:391–401, 2009.

[50] A. Davenport and J. Kalagnanam. A computational study of the Kemeny rule for preference aggregation. In *AAAI '04: Proceedings of the 19th National Conf. on Artificial Intelligence*, pages 697–702, 2004.

[51] P. Diaconis. *Group Representation in Probability and Statistics.* Institute of Mathematical Statistics, 1988.

[52] J. Díaz, M. Serna, and D. M. Thilikos. Efficient algorithms for counting parameterized list H-colorings. *Journal of Computer and System Sciences*, 74(5):919–937, 2008.

[53] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truss. Fixed-parameter tractability results for feedback set problems in tournaments. *J. of Discrete Algorithms*, 8(1):76–86, 2010.

[54] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[55] R. G. Downey and D. M. Thilikos. Confronting intractability via parameters. *Computer Science Review*, 5(4):279–317, 2011.

[56] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW '01: Proceedings of the 10th International Conference on World Wide Web*, pages 613–622, 2001.

[57] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. Manuscript, 2001.

[58] E. Ephrati and J. S. Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *AAAI '91: Proceedings of the 9th National Conference on Artificial Intelligence - Volume 1*, pages 173–178. AAAI Press, 1991.

[59] E. Ephrati and J. S. Rosenschein. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1-4):13–67, 1997.

[60] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28:652–673, 1999.

[61] P. Erdös and R. Rado. Intersection theorems for systems of sets. *Journal London Math. Soc.*, 35:85–90, 1960.

[62] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *PODS '04: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 47–58, 2004.

[63] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.

[64] U. Feige. Faster FAST (feedback arc set in tournaments). *CoRR*, abs/0911.5094, 2009.

[65] H. Fernau. On parameterized enumeration. In Oscar Ibarra and Louxin Zhang, editors, *Computing and Combinatorics*, volume 2387 of *Lecture Notes in Computer Science*, pages 151–179. Springer Berlin / Heidelberg, 2002.

[66] H. Fernau. Parameterized algorithmics: A graph-theoretic approach. *Germany: Habilitationsschrift, Universität Tübingen*, 2005.

[67] H. Fernau. Edge dominating set: efficient enumeration-based exact algorithms. In *IWPEC '06: Proceedings of the 2nd International Workshop on Parameterized and Exact Computation*, Lecture Notes in Computer Science, pages 142–153, Berlin, 2006. Springer.

[68] H. Fernau, F. V. Fomin, D. Lokshtanov, M. Mnich, G. Philip, and S. Saurabh. Ranking and drawing in subexponential time. In *IWOCA '10: Proceedings of the 21st International Conference on Combinatorial Algorithms*, pages 337–348, Berlin, Heidelberg, 2011. Springer-Verlag.

[69] H. Fernau and D. F. Manlove. Vertex and edge covers with clustering properties: Complexity and algorithms. *J. of Discrete Algorithms*, 7(2):149–167, 2009.

[70] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. In *ICDT '01: Proceedings of the 8th International Conference on Database Theory*, pages 22–38, 2001.

[71] J. Flum and M. Grohe. The parameterized complexity of counting problems. *SIAM J. on Computing*, 33:892–922, 2004.

[72] F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov. On two techniques of combining branching and treewidth. *Algorithmica*, 54:181–207, 2009.

[73] M. Frances and A. Litman. On covering problems of codes. *Theory Comput. Syst.*, 30:113–119, 1997.

[74] M. Frick. Generalized model-checking over locally tree-decomposable classes. In *STACS '02: Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*, pages 632–644, London, UK, UK, 2002. Springer-Verlag.

[75] K. Fukuda, T. M. Liebling, and F. Margot. Analysis of backtrack algorithms for listing all vertices and all faces of a convex polyhedron. *Comput. Geom. Theory Appl.*, 8(1):1–12, 1997.

[76] K. Fukuda and T. Matsui. Finding all the perfect matchings in bipartite graphs. *Applied Mathematics Letters*, 7(1):15–18, 1994.

[77] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, 1979.

[78] L. Gąsieniec, J. Jansson, and A. Lingas. Efficient approximation algorithms for the hamming center problem. In *SODA '99: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 905–906, Philadelphia, 1999. SIAM.

[79] O. Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[80] J. Gramm and R. Niedermeier. A fixed-parameter algorithm for minimum quartet inconsistency. *J. Comput. Syst. Sci.*, 67(4):723–741, 2003.

[81] J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for closest string and related problems. In P. Eades and T. Takaoka, editors, *ISAAC '01: Proceedings of the 12th International Symposium on Algorithms and Computation*, volume 2223 of *LNCS*, pages 441–453, Heidelberg, 2001. Springer.

[82] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.

[83] J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. *Theor. Comp. Sys.*, 41(3):501–520, 2007.

[84] H. W. Hamacher. *K* best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4:123–143, 1985.

[85] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[86] H. Ito and K. Iwama. Enumeration of isolated cliques and pseudo-cliques. *ACM Trans. Algorithms*, 5(4):40:1–40:21, 2009.

[87] B. N. Jackson, P. S. Schnable, and S. Aluru. Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 5(2):161–171, 2008.

[88] D. S. Johnson and C. H. Papadimitriou. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[89] R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

[90] H. Kaplan, R. Shamir, and R. E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28(5):1906–1922, 1999.

[91] M. Karpinski and W. Schudy. Faster algorithms for feedback arc set tournament, Kemeny rank aggregation and betweenness tournament. In O. Cheong, K. Chwa, and K. Park, editors, *ISAAC '10: Proceedings of the 21st International Symposium on Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 3–14. Springer Berlin Heidelberg, 2010.

[92] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:575–591, 1959.

[93] J. G. Kemeny and J. Laurie Snell. *Mathematical Models in the Social Sciences.* The MIT Press, 1973.

[94] C. Kenyon-Mathieu and W. Schudy. How to rank with few errors. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.

[95] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38–40):3640–3654, 2009.

[96] A. V. Kostochka. A bound of the cardinality of families not containing Δ-systems. *The Mathematics of Paul Erdös*, pages 229–235, 1996.

[97] A. V. Kostochka. An intersection theorem for systems of sets. *Random Struct. Algorithms*, 9:213–221, 1996.

[98] A. V. Kostochka, V. Rödl, and L. A. Talysheva. On systems of small sets with no large delta-subsystems. *Comb. Prob. and Comp*, 8:265–268, 1999.

[99] J. C. Lagarias. Point lattices. In R. L. Graham, M. Grötschel, and L. Lovász, editors, *Handbook of combinatorics (vol. 1)*, pages 919–966. MIT Press, Cambridge, MA, USA, 1995.

[100] J. K. Lanctot, M. Li, B. Ma, S. Wang, and L. Zhang. Distinguishing string selection problems. *Inform. and Comput.*, 185(1):41–55, 2003.

[101] E. L. Lawler. A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.

[102] H. W. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.

[103] M. Li, B. Ma, and L. Wang. Finding similar regions in many strings. In *STOC '99: Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 473–482, New York, 1999. ACM.

[104] D. Lokshtanov, D. Marx, and S. Saurabh. Slightly superexponential parameterized problems. In *SODA '11: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–776, San Francisco, 2011. SIAM.

[105] B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009.

[106] M. Mahajan and V. Raman. Parameterizing above guaranteed values: MaxSat and MaxCut. *Journal of Algorithms*, 31(2):335–354, 1999.

[107] M. Mahajan, V. Raman, and S. Sikdar. Parameterizing above or below guaranteed values. *J. Comput. Syst. Sci.*, 75(2):137–153, 2009.

[108] D. Marx. Parameterized complexity of constraint satisfaction problems. *Comput. Complex.*, 14(2):153–183, 2005.

[109] C. McCartin. Parameterized counting problems. In K. Diks and W. Rytter, editors, *MFCS '02: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, volume 2420 of *Lecture Notes in Computer Science*, pages 556–567. Springer-Verlag, 2002.

[110] B. Monjardet. Tournois et ordres médians pour une opinion. *Mathématiques et Sciences humaines*, pages 55–73, 1973.

[111] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28, 1965.

[112] K. G. Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations Research*, 16(3):682–687, 1968.

[113] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[114] R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *J. Algorithms*, 47:63–77, 2003.

[115] N. Nishimura, P. Ragde, and D. M. Thilikos. Parameterized counting algorithms for general graph covering problems. In *WADS '05: Proceedings of the 9th International Conference on Algorithms and Data Structures*, pages 99–109, Berlin, Heidelberg, 2005. Springer-Verlag.

[116] N. Nishimura and N. Simjour. Enumerating neighbour and closest strings. In *IPEC '12: Proceedings of the 7th International Symposium on Parameterized and Exact Computation*, pages 252–263, 2012.

[117] N. Nishimura and N. Simjour. Parameterized enumeration of (locally-) optimal aggregations. In *WADS '13: Proceedings of the 14th International Workshop on Algorithms and Data Structures*, page to appear, 2013.

[118] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[119] C. Paul, A. Perez, and S. Thomassé. Conflict packing yields linear vertex-kernels for k-FAST, k-dense RTI and a related problem. In *MFCS '11: Proceedings of the 36th International Conference on Mathematical Foundations of Computer Science*, pages 497–507, 2011.

[120] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[121] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5:237–252, 1975.

[122] N. Robertson and P. D. Seymour. Graph minors. III. Planar tree-width. *J. of Comb. Theory Ser. B*, 36(1):49–64, 1984.

[123] N. Simjour. Improved parameterized algorithms for the Kemeny aggregation problem. In *IWPEC '09: Proceedings of the 4th International Workshop on Parameterized and Exact Computation*, pages 312–323, 2009.

[124] C. Sloper and J. A. Telle. An overview of techniques for designing parameterized algorithms. *The Computer Journal*, 51(1), 2008.

[125] R. Stearns. The voting problem. *The American Mathematical Monthly*, 66(9):761–763, 1959.

[126] N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller. A linear-time algorithm for the 1-mismatch problem. In F. Dehne, A. Rau-Chaplin, J. Sack, and

R. Tamassia, editors, *WADS '97: Proceedings of the 5th International Workshop on Algorithms and Data Structures*, volume 1272 of *LNCS*, pages 126–135, Heidelberg, 1997. Springer.

[127] M. Thurley. Kernelizations for parameterized counting problems. In *TAMC '07: Proceedings of the 4th International Conference on Theory and Applications of Models of Computation*, pages 705–714, Berlin, Heidelberg, 2007. Springer-Verlag.

[128] M. Truchon. *An extension of the Condorcet criterion and Kemeny orders*. Cahier 98–15 du Centre de Recherche en Economie et Finance Appliquees, 1998.

[129] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[130] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. on Computing*, 8:410–421, 1979.

[131] J. van Rooij and H. Bodlaender. Exact algorithms for edge domination. *Algorithmica*, pages 1–29, 2011.

[132] J. M. M. Van Rooij and H. L. Bodlaender. Exact algorithms for edge domination. In *IWPEC '08: Proceedings of the 3rd International Workshop on Parameterized and Exact Computation*, pages 214–225, Berlin, Heidelberg, 2008. Springer-Verlag.

[133] A. Van Zuylen and D. P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Mathematics of Operations Research*, 34(3):594–620, 2009.

[134] J. Wang, B. Chen, Q. Feng, and J. Chen. An efficient fixed-parameter enumeration algorithm for weighted edge dominating set. In *FAW '09: Proceedings of the 3rd International Workshop on Frontiers in Algorithmics*, pages 237–250, 2009.

[135] J. Wang and G. Jiang. A fixed-parameter enumeration algorithm for the weighted FVS problem. In *TAMC '09: Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation*, pages 390–399, Berlin, Heidelberg, 2009. Springer-Verlag.

[136] L. Wang and B. Zhu. Efficient algorithms for the closest string and distinguishing string selection problems. In X. Deng, J. Hopcroft, and J. Xue, editors, *FAW '09: Proceedings of the 3rd International Workshop on Frontiers in Algorithmics*, volume 5598 of *LNCS*, pages 261–270, Heidelberg, 2009. Springer.

[137] V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC '12: Proceedings of the 44th Annual ACM Symposium on Theory of Computing*, pages 887–898. ACM, 2012.

[138] M. Xiao and J. Guo. A quadratic vertex kernel for feedback arc set in bipartite tournaments. In *MFCS '12: Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, pages 825–835, 2012.

[139] H. P. Young. Condorcet's theory of voting. *The American Political Science Review*, 82(4):1231–1244, 1988.

[140] H. P. Young and A. Levenglick. A consistent extension of condorcet's election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

[141] R. Zhao and N. Zhang. A more efficient closest string algorithm. In *BICoB '10: Proceedings of the 2nd International Conference on Bioinformatics and Computational Biology*, pages 210–215, 2010.