Ranking and Clustering with Advice

Tom Coleman

September 2009

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

Produced on archival quality paper

Department of Computer Science and Software Engineering,
The University of Melbourne,
Victoria, Australia.

Abstract

There is a wide range of combinatorial optimisation problems which provide us with local, pairwise advice about data points, and then ask us to form a global *clustering* or *ranking* of those points. The objective is to maximise the amount of advice respected, or minimise the amount ignored. In this thesis, we study such problems from approximation and heuristic perspectives, as exact solutions are unlikely, given each problem is NP-complete.

In Chapter 3 we explore the method of *relaxing* a combinatorial problem to a continuous problem, and apply it to find a solution to the problem of Affinity Clustering with Advice. We show how two well-known relaxation techniques for Affinity Clustering problems, namely Spectral Clustering and Semi-Definite Programming, can be extended in a sensible way to incorporate (possibly inconsistent) advice. We perform experiments to demonstrate that our techniques do in fact out-perform techniques which ignore such advice, or treat it in a simplistic fashion.

In Chapter 4 we consider local-search solutions to Min Feedback Arc Set and Min 2-Correlation Clustering, two fundamental advice-based problems. We consider both simple local-search approaches as well as more complicated algorithms which are motivated by that simple approach. In the case of Min Feedback Arc Set, we perform a thorough survey of existing algorithms on both synthetic and non-synthetic data-sets and find that our algorithms, when combined with a powerful existing local-search techniques, are as strong, or stronger

than any other technique. In the case of Min 2-Correlation Clustering, we not only find that our algorithm, which uses a local-search step, out-performs all existing algorithms, but additionally that it is a guaranteed 2-approximator. This result is of particular interest as few local-search algorithms have (known) approximation guarantees; in fact we demonstrate many counter-examples throughout this thesis.

Finally, in Chapter 5, we consider the problem of k-Consensus Clustering, which asks us to form a representative k-clustering of many input clusterings. We are interested in this restriction of Consensus Clustering as the unrestricted problem has recently been shown to be APX-hard for minimisation. Indeed, we complement that result by providing a polynomial-time approximation scheme for both maximisation and minimisation of k-Consensus Clustering, relying on a connection to the k-Cut problems.

Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,

(ii) due acknowledgment has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Signed,

Tom Coleman
18th September 2009

Preface

The original work in this thesis is related to the following four papers, three of which have been peer reviewed and appeared in conference proceedings, and fourth of which is under review:

- Coleman, T., and Wirth, A. A Polynomial Time Approximation Scheme for k-Consensus Clustering. In *Proceedings of the Twenty-First ACM-SIAM Symposium on Discrete Algorithms* (2010), to appear.
  - The content of this paper forms the original work of Chapter 5.
  - The metric max-k-Correlation Clustering PTAS of Section 5.3 appeared first in this paper.
  - The metric min-k-Correlation Clustering PTAS of Section 5.4 appeared first in this paper.
- Coleman, T., Saunderson, J., and Wirth, A. A local-search 2-approximation for 2-correlation clustering. In *Proceedings of the Sixteenth Annual European Symposium on Algorithms* (2008), pp. 308-319.
  - The content of this paper appears as the original work on Min 2-Correlation Clustering presented in Chapter 4.
  - The algorithms presented in Section 4.3.1 appeared first in this paper.
  - The experimental results in Section 4.3.2 appeared first in this paper.

  - The counter-examples and proof that PASTA-TOSS is a 2-approximation in Section 4.3.3 appeared first in this paper.

- COLEMAN, T., SAUNDERSON, J., AND WIRTH, A. Spectral Clustering with Inconsistent Advice. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning* (2008), pp.152-159.
  - The content of this paper forms the original work of Chapter 3.
  - The construction of the spectral clustering algorithms constrained by inconsistent advice (named METHODS ONE, TWO AND THREE) outlined in Section 3.6 appeared first in this paper.
  - The experiments on these algorithms in Section 3.7 appeared first in this paper.

- COLEMAN, T., AND WIRTH, A. Ranking tournaments: Local search and a new algorithm. In *Proceedings of the Ninth workshop on Algorithm Engineering and Experiments* (2008), pp.133-141. Also, appeared in *Journal of Experimental Algorithmics*, Volume 14 (2009), pp. 2.6-2.22.
  - The content of this paper appears as the original work on MIN FEEDBACK ARC SET presented in Chapter 4.
  - The algorithms presented in Section 4.2.1 appeared first in this paper.
  - The experimental results from Section 4.2.2 appeared first in this paper.
  - The counter-examples in Section 4.2.3 appeared first in this paper.

Acknowledgments

First of all I would like to wholeheartedly thank Tony Wirth for such a fine job of supervising me. If he hadn't 'saved' me from the mathematics department by tempting me with such interesting algorithmic problems, I am sure I would not be writing this thesis today. His important insights, excellent writing advice, and attention to detail have complemented my shortcomings in those areas. He always made time to discuss not only my work, but also, importantly, whatever else came to mind, whether it be world events, sporting news or the latest Apple product.

Many thanks to Peter Stuckey for keeping me going in the right direction throughout my studies, and especially for making the time to read my thesis in my time of need. Also, thanks to Adrian Pearce and Chris Leckie for words of advice and encouragement.

Many thanks to Laurence Park for providing the dataset for the **WebCommunities** set of experiments. Thanks to Compaq for the **EachMovie** dataset, and to UC Irvine for the UCI repository. Thanks to Ian Davidson for encouraging us to work on the spectral clustering problem and for hosting me at UC Davis, and to the Department of Computer Science and Software Engineering, the Australian Research Council and the School of Engineering for funding my conference attendances.

Thanks to all my family and friends for putting up with me throughout my studies; but in particular, my parents, Helen Measday and Edwin Coleman deserve particular thanks for not only encouraging me from the very beginning, but repeatedly reading and re-reading proofs of every academic document I have ever produced. Also, special thanks to my girlfriend, Sophie Oldfield, for supporting me in every way possible, and being excited and proud for me, even though I consistently failed to properly keep her informed about my

progress.

Finally I would like to thank the Stella Mary Langford Scholarship Fund for supporting my thesis with the Stella Mary Langford Scholarship.

Contents

List of Figures

List of Tables

Notation

Throughout G refers to some kind of graph over a set of vertices V, sometime referred to as points. The symbol $n = |V|$ is reserved for the total number of vertices in the graph. In a clustering, k always refers to the number of clusters.

The notation $[\ell]$ means the set $\{1, 2, \ldots, \ell\}$, and $[\ell]_0 = \{0, 1, 2, \ldots, \ell\}$. $\mathbb{R}$ refers to the reals, $\mathbb{R}^+$ to the positive reals, and $\mathbb{R}_0^+$ to the non-negative reals. The notation

$$\big|\text{object} : \text{condition}\big|,$$

is shorthand for

$$\big|\{\text{object} : \text{condition}\}\big|.$$

Chapter 1

Introduction

In this thesis, we study algorithms for combinatorial optimisation problems. We investigate these algorithms with two orthogonal perspectives; both *practical* and *theoretical*. The intersection of those two perspectives is of particular interest. The problems we study are *graph problems*; we are interested in two flavours in particular: problems which ask us to *cluster* vertices and problems which ask us to *rank* them.

An optimisation problem has an objective function: this is a measure of the quality of any feasible solution to the problem—we seek to find the solution which either *maximises* or *minimises* that objective. The problems that we study in this thesis are exclusively NP-complete—that means that we cannot hope define a method that always finds the optimal solution in a polynomial amount of time. With this difficulty in mind, there are two approaches that we consider in order to solve such problems well.

The theoretical approach seeks to find algorithms which are *provably* good approximators. This means that even for the hardest possible instance of the problem, the algorithms will never do significantly worse than the optimal solution. Moreover, they accomplish this whilst still operating in time polynomial in the input size. Such algorithms are interesting from a mathematical perspective; additionally they give practitioners confidence in the solutions that they generate. A very famous example is the Max-Cut SDP of Goemans and Williamson [54], which is guaranteed to find a solution with objective value within a factor of 0.878 of the best possible.

The practical approach focuses on the problem instances that occur commonly

in practice. This values algorithms for their ability to produce high quality solutions to such instances in reasonable amounts of time. Thus we seek to find algorithms which are *fast* and *effective*, of obvious import to those who are solving the problem in real world situations.

In this thesis, we consider ranking and clustering graph problems from both of the above perspectives. Additionally, we try to investigate the middle ground between the two approaches. We test algorithms which have previously been considered only from theoretical perspectives; we prove theoretical results about algorithms that were only ever used as heuristics. In this way, we can say which algorithm is 'best' for a given problem.

1.1 Optimisation

The study of algorithms is interesting precisely because so many problems that are simple to describe are so difficult to solve. A large class of problems—the NP-complete problems—are widely believed to be insolvable; at least, no algorithm is likely to exist that will find the optimal solution in reasonable time in all cases. In this work we study algorithms that attempt to circumvent this difficulty by finding solutions in polynomial time that, although not optimal, are approximations to the optimal solution.

Although some problems of interest to computer scientists have a single 'correct' answer, many have a set of feasible solutions, of which one or more are *optimal*. To be able to 'approximate' the optimal solution, we need to have some measure by which we can judge the quality of any feasible solution. Once this criterion of quality is established, we can compare the quality of some sub-optimal feasible solution to the optimum in some meaningful way.

For example, in a scheduling problem, the quality of a solution might be measured by the total time taken to complete all the jobs in the schedule. Although there are many feasible schedules to complete all the jobs, there may only be one that does so in the minimum amount of time. An approximate schedule may take a little longer than that optimal schedule; we search for a solution does not take *too much* longer than that optimal schedule. Also, we must ensure than calculating our schedule does not take more than an amount of time polynomial in the input size.

Restricting algorithms to polynomial running times is a standard approach to ensuring that algorithms are practical for reasonably sized problem examples. Although the super-polynomial algorithms are an alternate approach to avoiding NP-completeness (see Section 1.5.3), we will not consider such approaches in this thesis.

A problem which admits such a quality criterion is called an **Optimisation Problem.** An optimisation problem is either a *maximisation* problem or a *minimisation* problem. We are interested in problems of both types, although it is often the case that minimisation is harder (from an approximation perspective). Mathematically:

Problem: Optimisation Problem

An optimisation problem $\mathcal{P}$ consists of:

- A set of instances $\mathcal{I}$;
- For each $I \in \mathcal{I}$, a set of solutions to I, $\mathcal{S}(I)$; and
- An eponymous objective function $\mathcal{P} : \mathcal{I} \times \mathcal{S}(I) \to \mathbb{R}_0^+$

So $\mathcal{P}(I, S)$ measures the quality of S as a solution to I. We are asked:

Given: an instance $I \in \mathcal{I}$;
Find: an *optimal* solution $S_I^* \in \mathcal{S}(I)$, that is, a solution that maximises (or minimises) $\mathcal{P}(I, S_I^*)$.

As the specific instance that we are considering is usually fixed, we often abuse notation and write $\mathcal{P}(S)$ rather than $\mathcal{P}(I, S)$; we also often write $\mathcal{P}^* = \mathcal{P}(I, S_I^*)$, the cost of the optimal solution. A diagrammatic version of that definition is illustrated in Figure 1.1.

For example, consider the Max-Cut problem, one of the simplest *graph problems*. Max-Cut asks us to take a graph and split it into two pieces, maximising the following objective:

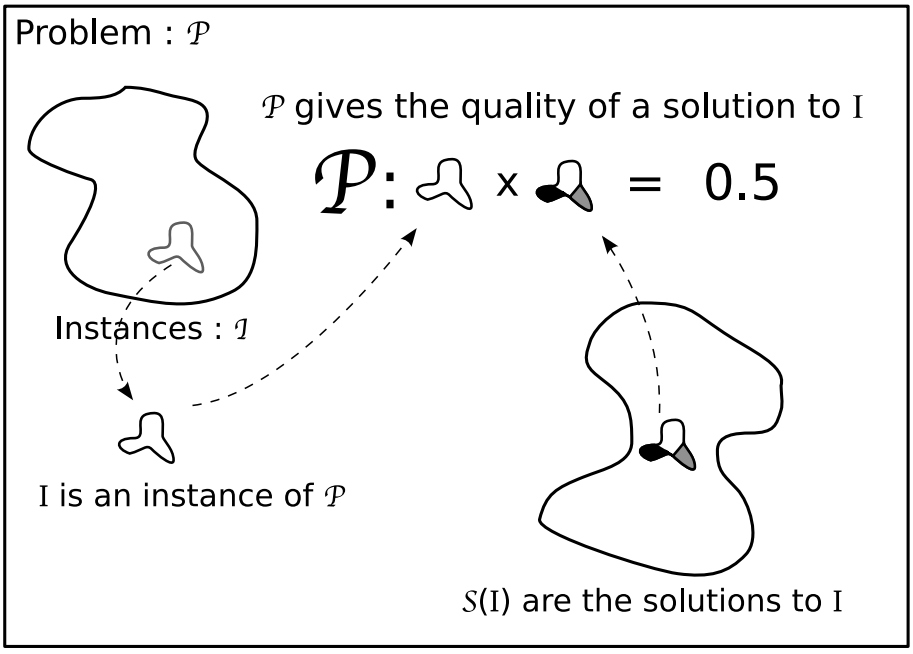


Figure 1.1: An optimisation problem $\mathcal{P}$ has many instances $\mathcal{I}$; $\mathcal{P}$ measures the quality of a solution $S \in \mathcal{S}(I)$ to an instance $I \in \mathcal{I}$.

Problem: Max-Cut

> Given: $G = (V, E) \in \mathcal{I}$, a graph;
> Find: A partition $S = (S_1, S_2)$ of V into two subsets. We seek to find a partition to maximise the value of:

$$\textsc{Max-Cut}(G, S) = \big|(u, v) \in E : u \in S_1, v \in S_2\big|$$

So Max-Cut is a maximisation problem. Like many maximisation problems, Max-Cut has a complementary minimisation problem, called Min-Uncut. We say these two problems are complementary as they essentially are aiming to do the same thing—they just measure the degree of success in a complementary fashion. Min-Uncut again asks us to split a graph into two pieces whilst maximising the number of edges cut. But in Min-Uncut, rather than directly counting the edges cut, we measure the number of edges that we *do not cut*:

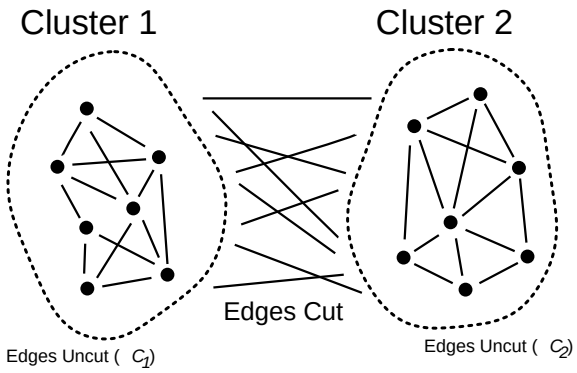


Figure 1.2: Demonstration of the MAX-CUT/MIN-UNCUT cost of a 2-clustering (C_1, C_2). As any edge in E must either be cut or uncut, we can see that MAX-CUT(G, S) + MIN-UNCUT$(G, S) = |E|$, for any solution S.

Problem: MIN-UNCUT

GIVEN: $G = (V, E) \in \mathcal{I}$, a graph;
FIND: A partition $S = (S_1, S_2)$ of V into two subsets, to minimise:

$$\text{MIN-UNCUT}(G, S) = \left|(u, v) \in E : u, v \in S_1 \text{ or } u, v \in S_2\right|$$

Figure 1.2 demonstrates that for any G and S,

$$\text{MAX-CUT}(G, S) + \text{MIN-UNCUT}(G, S) = |E|\,. \tag{1.1}$$

So a solution S_I^* that maximises MAX-CUT for some given G will minimise MIN-UNCUT for G. That means that for any instance I, the two problems share optimal solutions. However, a good approximation to one problem may not be a good approximation to another.

Consider Figure 1.3. Here we have two potential solutions to a MAX-CUT problem—the optimal solution, with MAX-CUT value 5, and MIN-UNCUT cost 0, is well approximated for MAX-CUT by the approximation, which has MAX-CUT value 4, 4/5 of the optimum. However, in terms of MIN-UNCUT, the approximation ratio is not even defined. So we can see that—at least multiplicatively—a good algorithm for maximisation will not be a good algorithm for minimisation.

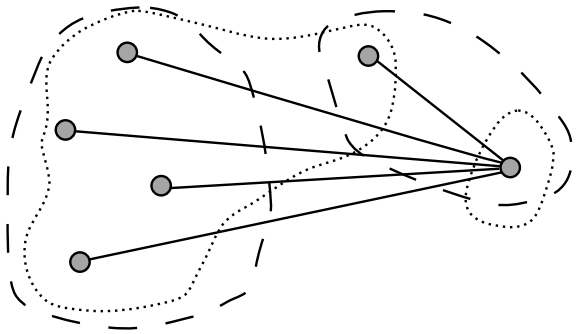


Figure 1.3: An example demonstrating that a good maximisation approximation is not necessarily a good minimisation approximation. Here the optimal solution is the dotted clustering—with a Max-Cut value of 5 and a Min-Uncut cost of 0. The dashed clustering is a good approximation to Max-Cut—it cuts 4 of the 5 edges—however as it leaves 1 edge uncut, it is (multiplicatively) far worse a solution to Min-Uncut.

1.2 Approximation

The Max-Cut example discussed above emphasises exactly what kind of approximate solutions we aim to find for our optimisation problems. We aim to find solutions that are within some *fraction* of the optimum, in terms of the objective value. This is the standard measure of how well an algorithm approximates a problem, and is the same formulation used in the very first approximation results of Garey et al. [51] and Johnson [61].

Let us be completely clear about what we mean by an approximation factor, α:

Definition 1. *A solution S to an instance I of a maximisation problem $\mathcal{P}$ is a α-approximate solution if*

$$\mathcal{P}(I, S) \geq \alpha \mathcal{P}(I, S_I^*)$$

Similarly, if $\mathcal{P}$ is a minimisation problem, S is α-approximate if

$$\mathcal{P}(I, S) \leq \alpha \mathcal{P}(I, S^*)$$

For a minimisation problem, α will be greater than 1—a better solution has a smaller α value. Indeed, later, we will see that finding solutions which are α-approximations where α is arbitrarily close to one forms an important arm of re-

search in theoretical approximation algorithms.

We can approach this problem from two orthogonal directions—theoretical and practical. Theoreticians try to design algorithms with guarantees that they will always find good approximate solutions, running in time polynomial in the problem size—although this doesn't always practically translate into fast, effective algorithms. Algorithm engineers don't worry about approximation *guarantees*—instead their algorithm are rated by how well they *tend* to perform (in terms of approximation), and how long they take to execute.

1.3 Theoretical Guarantees

We now have a concept of an approximately good solution to a given instance of an optimisation problem. We now turn our attention to designing algorithms which will *always* find such solutions in a (theoretically) reasonable amount of time. If we have an algorithm for which we can prove that we will always come within some fraction of the optimal solution, then we can feel comfortable using it, secure in the knowledge that we can only do so much worse than the optimum.

We define an α-approximation algorithm as one which always finds an α-approximate solution. Formally:

Definition 2. *An algorithm $\mathcal{A}$ for a problem $\mathcal{P}$ is an α-approximation algorithm if, for all instances I, $\mathcal{A}(I)$ is an α-approximate solution.*

For example, consider the following simple algorithm for the Max-Cut problem, which runs in $O(n)$ time:

Algorithm: Random Partitioning

> Given an a graph $G = (V, E) \in \mathcal{I}$, let $S = (S_1, S_2)$ be a random 2-clustering—that is, for each $v \in V$, let v be in S_1 with probability $1/2$, and S_2 otherwise.

Although the Random Partitioning algorithm seems fairly unmotivated, the structure of the problem means that in fact we can prove something about it, on average:

Remark 1. Random Partitioning *is an expected* 1/2*-approximation algorithm for the* Max-Cut *problem.*

Proof. Given a input graph $G = (V, E)$, consider any edge $e \in E$. Let $S = (S_1, S_2) =$ Random Partitioning(G). Is e cut by S? The probability that e is cut is exactly $1/2$. So the expected contribution to the Max-Cut objective, by e, is thus:

$$\mathbb{E}(e\text{'s contribution}) = 1/2$$

So, by linearity of expectation,

$$\mathbb{E}(\text{Max-Cut}(G, S)) = \sum_{e \in E} \mathbb{E}(e\text{'s contribution}) = 1/2|E| \geq 1/2\,\text{Max-Cut}^*$$

□

Note that to complete this proof we needed to use an upper bound on the value of Max-Cut*—namely that we can at most cut all of the edges of G—so Max-Cut$^* \leq |E|$. Usually such an upper bound is needed to prove approximation guarantees.

To emphasise the point about the difference between maximisation and minimisation, note that for the Min-Uncut problem, Random Clustering is not an α-approximation algorithm for any α. The example from Figure 1.3 demonstrates this: the expected Min-Uncut cost of a Random Clustering is 5/2, whereas the optimal cost is 0. In fact, unless the algorithm happens to pick the correct solution (which will happen with probability $\frac{1}{2^4} = 0.0625$), it will output a solution with positive cost, which is infinitely worse than the optimum.

1.3.1 Polynomial Time Approximation Schemes (PTASes)

In the previous section, we discussed avoiding the NP-completeness problem by designing algorithms which manage to run in polynomial time by not attempting to find an optimal solution, but instead finding α-approximate solutions. In some cases, it is possible to improve the approximation factor of an algorithm with a corresponding increase in running time. Thus we can trade off running time for better quality solutions.

If it is possible to to improve the algorithm to an arbitrarily close approximation factor to 1, whilst maintaining running times polynomial in the size of I, then we have a Polynomial Time Approximation Scheme (PTAS).

Definition 3. *A set of algorithms ($\mathcal{A}_\epsilon$) forms a PTAS for a maximisation problem $\mathcal{P}$, if, for each $\epsilon > 0$, $\mathcal{A}_\epsilon$ runs in polynomial time (in the size of I—not necessarily ϵ) and is a $(1-\epsilon)$-approximation algorithm. For minimisation, each $\mathcal{A}_\epsilon$ must be a $(1+\epsilon)$-approximation algorithm.*

So a problem that admits a PTAS is in one sense 'solvable'. We have—theoretically—done as well as possible to overcome the intrinsic NP-completeness of the problem.

One point that should be stressed is that although each individual algorithm $\mathcal{A}_\epsilon$ is polynomial in running time, in terms of n, there is no restriction on running time in terms of ϵ. So for instance, the running time can be $O(n^{1/\epsilon})$, which can be quite impractical, even for relatively large values of ϵ.

As we have stated previously, the problems we consider are all members of the set NP—the problems solvable non-deterministically in polynomial time. In fact, each problem is NP-complete—as hard as any problem in NP. In fact, we'll see that most of the problems are also in another, more restrictive class: the APX problems. These are problems which admit constant factor approximation algorithms—α-approximation algorithms, where α does not depend on n.

It is known[7] that if P$\neq$ NP, then APX$\neq$ *PTAS*—the set of problems that admit PTASes. Thus any problem that is APX-hard—as hard as any problem in APX—cannot admit a PTAS. Although we will not directly prove it for any problems, in this thesis we will encounter problems in this work which are known to be APX-hard—these problems we know will not admit a PTAS.

1.4 Practical Interests

Although having theoretical guarantees about the performance of an algorithm is reassuring for a practitioner, ultimately, it is the performance of the algorithm on instances of the problem that are typical of their application that is their main

concern. Additionally, worst-case running time is sometimes less important than the average run-time of an algorithm.

Often approximation algorithms, and PTASes in general, make large sacrifices in running time—whilst still remaining polynomial—in order to achieve better worst-case performance bounds. Effectively this can mean sacrificing performance on most instances in order to achieve a better approximation result on a very few 'difficult' instances.

For example, one PTAS (for the Correlation Clustering problem), which we will see later, requires taking a sample and trying all possible clusterings of that sample. Although the sample has constant size, the constant is greater than $4,000$, leading to a need to try more than $2^{4,000}$ different combinations—a completely unreasonable proposition. Here the PTAS becomes a purely theoretical object—in this case it is not even feasible to run it for very small instances.

So the practical approach is a completely different (and often contradictory) perspective through which to view algorithms. Whilst studying algorithms from a theoretical perspective is rigorous, practical algorithm research is more of an experimental science, or engineering problem. Here we aim to design algorithms to solve a given problem for some set of typical instances, then experimentally validate the performance of those algorithms as opposed to the existing approaches.

When designing algorithms for optimisation problems, there can be situations where we can sacrifice running time for increased effectiveness. This can be similar to the idea that led to PTASes. Or we may be able to tweak some algorithm in a way which will tend to make it run faster, whilst degrading the performance. From a practical perspective, it is often very difficult to say what is the correct decision to make—such decisions are going to depend heavily on the specific application.

Also, it is a common (and under-reported) fact that algorithms which are more effective for a given problem can often take longer to execute. When you consider that algorithms may often be tuned in order to alter this running time/effectiveness trade-off, we need to be careful in order to properly compare the performance of one algorithm to another.

1.5 The Intersection

A specific focus of this thesis is the intersection of the practical and theoretical worlds. The difference in style between the two approaches has perhaps led to a lack of communication between the two disciplines. We are interested in the intersection from two directions.

1.5.1 Heuristics with Guarantees

Although algorithms designed purely from a practical perspective (heuristics) are not designed to enjoy them, they can sometimes have theoretical guarantees. For example, Coppersmith et al.[26] have shown that the very simple Iterated Kendall algorithm for the Min Feedback Arc Set problem (a problem we will examine closely) is a 5-approximation. This is an algorithm that was originally designed as a heuristic [68]—we will see later than it is a reasonably strong performer practically. The addition of an approximation guarantee to its list of positive points is of great value.

In another example, Ostrovsky et al. [86] demonstrate a variant of the classic Lloyd's algorithm for the k-means problem that is a PTAS, given a reasonable separation criterion on the input dataset. As variants of Lloyd's algorithm are ubiquitous in practical applications, yet previously had little theoretical justification, this result is very compelling.

A heuristic with an approximation guarantee will have much greater value than a purely theoretical algorithm, which may be completely unmotivated from a performance perspective, and so—often—will perform very poorly in practice.

1.5.2 Effective Approximation Algorithms

An approximation algorithm that has been designed from a purely theoretical perspective certainly could be effective in practice. Even though such an algorithm may have never even been implemented, let alone tested on a realistic data-set, it is quite possible that the ideas that led to the algorithm being theoretically interesting may lead to the algorithm performing well.

The case of PTASes are especially interesting. As we stated earlier, they are

often designed with no thought at all for a practical implementation—such an implementation can be realistically impossible. However, it might be possible to adapt the fundamental idea of the PTAS to a more realistic algorithm. For example in the Correlation Clustering PTAS mentioned earlier, the concept of spending extra effort to start with a good solution on a small sample of the data, and extending it to a full solution could prove fruitful in the design of a heuristic. Changing the sample size, or perhaps just using a less (time) expensive algorithm on the sample—rather than enumerating all possible solutions—could help us to obtain an algorithm 'approximating' the PTAS, which may work in practical situations.

1.5.3 Super-Polynomial Running Times

In this thesis, we concentrate on sacrificing solution quality to achieve polynomial running times for NP-complete problems. However, an alternate approach is to sacrifice running time in order to achieve the optimal solution quality. Such algorithms have super-polynomial running times, but can still be of interest to researchers.

In a practical scenario where problem instances remain reasonably bounded, it may be feasible to run an algorithm that is super-polynomial in instance size. In such cases, a brute force naive algorithm which examines every feasible solution to the problem is often not the best optima-finding algorithm. A Dynamic Programming [16] or Branch and Bound [75] algorithm can often find the optimal solution in a reasonable if super-polynomial amount of time.

Theoretical algorithms also exist which find the optimal solution in super-polynomial time. Fixed-parameter algorithms [37] investigate specific sub-problems where some parameter of the problem instances is fixed to a small constant size. Such algorithms can be very useful when an application is—or tends to be—restricted to such bounded instances. This is similar to our restriction of the Consensus Clustering problem to k clusters in Chapter 5, although in this case we are placing a restriction on the problem *solution*, not the problem *instances*, and thus are considering a different problem, rather than a sub-problem.

We mention such algorithms for completeness—in this thesis we concentrate only on algorithms that run in polynomial time.

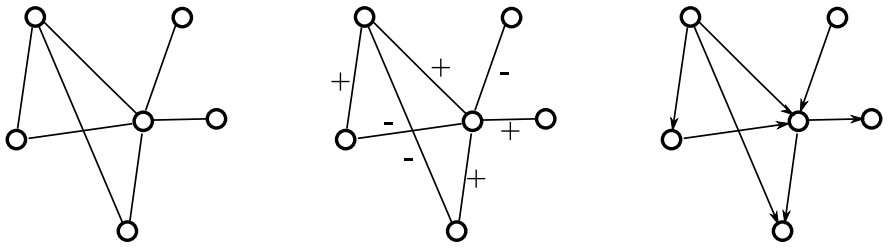
Chapter 2

Problems of Interest

Like the Max-Cut problem, each of the problems that we consider in this thesis is a graph problem; instances can always be represented as one or more graphs over a set of vertices V. Each problem asks us to find some kind of configuration of V; either a *clustering*—a grouping of V—or a *ranking*—an ordering of V.

The nature of graphs is to provide us with *pair-wise* information about points—they represent information about two vertices at a time. In this thesis, this information can be considered *advice*—we do not have to follow it, but it is best if we do. As we will see when we define the types of graphs we are interested in, the type of advice that they provide can be disparate. However, in all cases, they provide us with local information; in all cases the problems we are trying to solve ask us to find some global configuration of the data.

So the challenge that we have in each problem is to make global statements about V based only on piecemeal information. The difficulty in doing this in each case is the *inconsistency* of the pair-wise information—although we would like to respect each piece of advice that the graph gives us, in many instances this is not possible. The way in which this inconsistency is exhibited will vary with the type of graph involved, and with the exact problem. However, in every case, the question is how to choose which advice to respect, and which to ignore.



(a) Simple Graph. (b) Signed Graph. (c) Directed Graph.

Figure 2.1: Three of the types of graph that we consider.

2.1 Graph Problems

2.1.1 Simple Graphs

In the problem we have considered so far, Max-Cut, each instance I was a simple graph, the most basic instance that we will consider in this thesis. A simple graph $G = (V, E)$, consists of V, the set of vertices, and E, a set of unordered pairs of distinct vertices—the edges. So (u, v) and (v, u) are the same edge, and we are only allowed one in E—also, self-loops (u, u) are not allowed. Every problem in this thesis will have as instances simple graphs, or some generalisation of them.

One common generalisation of a simple graph is a weighted graph; a graph $G = (V, E, w)$ where $w : E \to \mathbb{R}_0^+$ is a weight function on the edges. Max-Cut can easily be extended to operate on a weighted graph; in this context, the Max-Cut value of a solution $S = (S_1, S_2)$ is given by

$$\textsc{Max-Cut}(G, S) = \sum_{e \in E, e \text{ cut}} w(e)$$

Most problems that whose objective depends on the number of edges in some configuration of a graph can be extended to a weighted graph in a similar way. The weight $w(e)$ of an edge indicates how important an edge is—for this reason, often it makes sense to normalise edge weights to be in the range $[0, 1]$. In this case, often a missing edge is equivalent to an edge of weight zero.

Given this definition, the simple, unweighted case can be seen as a special case of the the full weighted case—where all weights on existing edges are 1, and non-existent edges 0. For this reason, we sometimes refer to simple graphs as the 0/1

case.

2.1.2 Signed Graphs

In some problems we will consider, we need to think about two or more graphs over the same set of vertices. For instance, we may have one set of edges indicating which vertices should be separated (viz. the simple graph for the MAX-CUT problem), and other indicating which edges *should not*. It is convenient to represent both of these sets of advice with a single graph.

A signed graph is a simple graph with a labelling l on the edges, where $l : E \to \pm 1$. Thus, the set of $+$ edges could represent the edges that should not be cut, and the $-$ edges represent those that should. We define $E^+ = \{e \in E : l(e) = +1\}$, and E^- similarly.

2.1.3 Directed Graphs

Some problems are defined on directed graphs (digraphs). In these problems, rather than edges, E, we have arcs A, represented as $u \to v \in A$. An arc has a direction—directed from one vertex (u—the tail) to another (v—the head). The problems we will discuss on digraphs will be ordering problems. Note that digraphs can also have weights on the arcs, leading to weighted directed graphs.

One specific subset of these graphs of particular interest are the complete versions. Here complete means that for each pair $u, v \in V$, exactly one of $u \to v \in A$ or $v \to u \in A$. Complete directed graphs are referred to as *tournaments*.

2.1.4 Graph Properties

Certain properties of graphs are of particular interest.

Definition 4. *For a simple graph $G = (V, E)$, the* degree *of a vertex $v \in V$ is defined to be the number of edges incident to v. Or formally*

$$deg(v) = |u \in V : (u, v) \in E|$$

For a weighted graph, the degree is the sum of weights of incident edges. We can define $\operatorname{vol} S$, where $S \subset V$ is a subset of vertices, as the total degree of S, that is $\sum_{v \in S} \deg(v)$.

Definition 5. *For a directed graph, $G = (V, A)$, the* indegree *is the number of arcs directed into v:*

$$In(v) = |u \in V : u \to v \in A|$$

The outdegree *is the number of arcs directed out of v:*

$$Out(v) = |u \in V : v \to u \in A|$$

The indegree is sometimes known as the Kendall score. The indegree and outdegree of a weighted directed graph are defined analogously to that for a weighted simple graph.

The degree of a vertex in a signed graph is not usually important, however there are some neighbourhoods of interest:

Definition 6. *For a signed graph $G = (V, E, l)$, and a vertex v, we have*

$$N^+(v) = \{u \in V : (u, v) \in E^+\}$$

and

$$N^-(v) = \{u \in V : (u, v) \in E^-\}$$

Graph Cuts

If $S \subseteq V$ is a subset of vertices in a graph, and $\bar{S} = V \setminus S$ are the remaining vertices, then the quantity

$$\operatorname{cut}(S, \bar{S}) = \left|(u, v) \in E : u \in S, v \in \bar{S}\right|$$

counts the number of edges cut when separating S from V.

Definition 7. *If $S \subseteq V$ is a subset of vertices in a graph $G = (V, E)$, the induced graph $G|_S = (S, E|_S)$ is the graph over S containing only the edges $E|_S \subseteq E$ that have both endpoints in S. A similar definition holds for all of the graphs defined here.*

2.2 Clustering Problems

The first class of problems that we consider are *clustering* problems. These problems ask us to partition the set V into a family of clusters. In this thesis, we prefer a functional style, leading to the definition

Definition 8. *A* clustering *is a function $\mathcal{C} : V \to [n]$. So $\mathcal{C}(v)$ is the cluster that v belongs to.*

Usually a clustering function is not surjective; clustering n points into n clusters is not usually very interesting. In fact, many problems are trivial to solve when n clusters are allowed—for instance Max-Cut would place all vertices in singleton clusters to cut every edge. For these problems we introduce a restriction on the clustering function; we limit $\mathcal{C}$ to map to $[k]$, for some $k \leq n$. We usually refer to such problems as Max-k-Obj (or Min-k-Obj). Max-Cut is another name for Max-2-Cut, a special case of Max-k-Cut.

Some problems do not need the restriction on the number of clusters, but are of particular interest when that restriction is added.

The $k = 2$ case. When $k = 2$, rather than mapping V to $\{1, 2\}$, often we instead map V to ± 1.

Graph Cuts We label the edges cut by the a clustering $\mathcal{C}$:

$$E_c(\mathcal{C}) = \{(u, v) \in E : \mathcal{C}(u) \neq \mathcal{C}(v)\}.$$

We define the edges uncut, $E_u(\mathcal{C})$ similarly, and let $e_c(\mathcal{C}) = |E_c(\mathcal{C})|$, and $e_u(\mathcal{C}) = |E_u(\mathcal{C})|$.

We can represent a cluster more traditionally as a partition of V, into sets $C_1, \ldots, C_k$ (this is how we talked of Max-Cut above). We will sometimes abuse notation and write

$$\mathcal{C} = (C_1, \ldots, C_k) \text{ where } C_i = \{v \mid \mathcal{C}(v) = i\}.$$

We note that as we are *clustering* vertices and not *labelling* them, the choice of which particular cluster number gets given to which cluster is of no import. *Every*

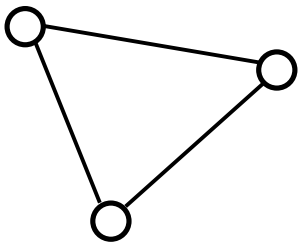


Figure 2.2: A difficulty for the MAX-CUT problem. As MAX-CUT can have at most 2 clusters, two of the vertices must be placed in the same cluster, thus not 'cutting' one of the three edges. Choosing which of the three to cut is the challenge for MAX-CUT.

objective function we consider is agnostic towards the choice of cluster number. We frequently refer to clustering functions which are strictly different, but lead to the same partition of V, as the same clustering.

2.2.1 What the graphs mean

In the clustering problem that we have considered so far, MAX-CUT, an edge (u, v) on G represents the information 'u should not be clustered with v'. This information can be considered *advice*—we aim to try and follow, or respect, as much of that advice as possible. In fact, the MAX-CUT objective function measures exactly how well we have followed the advice we have been given.

A difficultly arises when we have the additional information that 'v should not be clustered with x'—the edge (v, x)—and 'x should not be clustered with u'—the edge (x, u). In the MAX-CUT problem, where we are restricted to two clusters, there is no clustering of u, v, x that can satisfy all three of these edges. So the challenge for solving MAX-CUT is to break the correct edge (and choose which two points to place together). Figure 2.2 demonstrates the problem.

Another type of clustering problem considers an edge (u, v) to mean 'u *should* be clustered with v'. We call these problems AFFINITY CLUSTERING problems, as an edge between u and v indicates an affinity, or similarity, between u and v. In this case, there cannot be inherent contradictions in the input graph, and we need to add some more restrictions to the problem to make it interesting. We will consider such problems in Section 2.2.3.

A third set of clustering problems asks us to solve both of the types of problems considered above. So we have a *signed graph*, where we have both *must-link* advice (an edge labelled +1), and *cannot-link* advice. Although problems can be defined where such graphs represent *constraints* on the solution space—i.e. any feasible solution *must* respect the advice—in this thesis we only consider problems where the advice is *soft* and can be broken. The extent to which the advice is followed will invariably influence the objective function. We will consider such problems in Section 2.2.4.

2.2.2 Max-Cut Problems

Max-Cut restricts us to finding 2 clusters. We can define a more general version, where the number of clusters can be large (but is still specified—otherwise we could just put each point in its own cluster). So we have

Problem: Max-k-Cut

Given: A graph $G = (V, E)$;
Find: A clustering $\mathcal{C}$ into k clusters to maximise:

$$\text{Max-}k\text{-Cut}(G, \mathcal{C}) = |(u, v) \in E : \mathcal{C}(u) \neq \mathcal{C}(v)|$$

An extension to weighted graphs follows easily as it did for Max-Cut earlier. Similarly, we have

Problem: Min-k-Uncut

Given: A graph $G = (V, E)$;
Find: A clustering $\mathcal{C}$ into k clusters to minimise:

$$\text{Min-}k\text{-Uncut}(G, \mathcal{C}) = |(u, v) \in E : \mathcal{C}(u) = \mathcal{C}(v)|$$

A weighted version considers the total weight of uncut edges. Intuitively, this means $w(u, v)$ measures the extent to which we want to separate u and v. This intuition is confirmed by the sub-problem which will be very important in this thesis: metric-Min-k-Uncut, a special case of weighted-Min-k-Uncut, where the weights

form a *metric*. In this case, the $w(u, v)$ can be considered the *distance* between u and v. This is due to the definition:

Definition 9. *A weighted graph $G = (V, E, w)$ is a* metric *graph if w obeys the triangle inequality, that is, for all $u, v, x \in V$:*

$$w(u, v) \leq w(u, x) + w(x, v) \tag{2.1}$$

Previous Work

Max-Cut is an old and well-studied problem in computer science. In the general unweighted case, it is one of Karp's [66] original 21 NP-complete problems. Although there have been many approaches that have been attempted to solve the problem [91], the most famous approach is by Goemans and Williamson [54].

Goemans and Williamson's algorithm is a 0.878-approximation, which is achieved by relaxing the problem to a now classical semi-definite program. We will discuss this result in some detail in Section 3.2. Khot et al. [72] recently provided a reduction proving, assuming the Unique Games Conjecture [71], that this result is essentially the best possible. That is, given the conjecture, (which is generally assumed to be true), no algorithm is likely to achieve a better approximation for the general case. Raghavendra [92] extends this result to say something similar about SDP algorithms for many constraint satisfaction problems.

However, researchers have met with success by focusing on more specific cases of the Max-Cut problem. Of particular interest is the dense Max-Cut problem. A dense problem has $\Omega(n^2)$ edges in the unweighted case, or an average edge weight in $\Omega(n^2)$. A variety of results [42; 49; 6; 78] exist providing PTASes for variants of dense-Max-Cut and other related constraint satisfaction problems.

Of particular interest—and use to us later in this thesis—is the PTAS of Frieze and Kannan [49], which allows instances of weighted-Max-k-Cut which have *negative* edge weights. Although we did not allow negative weights in our definition of weighted graphs, this generalisation will be very useful in Section 5.3. They achieve this result by generalising Szemerédi's [100] regularity Lemma, a famous result from Graph Theory.

Fernandez de la Vega and Kenyon [44] then demonstrated how to extend a dense-

MAX-k-CUT PTAS to a metric-MAX-k-CUT PTAS. They construct a reduction between the problems, which duplicates vertices, favouring vertices of high degree—seen as more 'important' vertices. This means that when a dense-MAX-k-CUT PTAS samples the vertices, which they mostly do, it is more likely to choose one of these more important vertices.

Using this maximisation PTAS, Indyk [60] provided a PTAS for the complementary minimisation problem, metric-MIN-UNCUT. We will describe this PTAS in detail in Chapter 5, as well as the extension to the metric-MIN-k-UNCUT problem, which was provided by Fernandez de la Vega et al. [43].

2.2.3 AFFINITY CLUSTERING Problems: RCUT and NCUT

We now consider the case where an edge of G indicates that two vertices should be clustered together. Specifically, we focus on the weighted case, when each edge has weight, indicating the endpoints' mutual affinity. So we have pair-wise information, telling us to what extent various pairs need to be placed in the same cluster. However, as we have no reason to separate points, the simplest of AFFINITY CLUSTERING problems would have a very trivial solution: place all points in the same cluster. Obviously, such a clustering doesn't tell us very much about our data, and isn't really useful at all.

If we add a single cannot-link constraint, and force two specific points to be in different clusters—thus avoiding the all-in-one-cluster solution, we get the classical MIN-CUT problem, solvable in polynomial time by single-commodity flow algorithms [48]. However, there is certainly no guarantee that we can necessarily know which pair of points should be the ones forced into separate clusters. Additionally, it is quite possible that one of the points will be placed in an extremely small cluster (perhaps by itself), a situation which is not markedly better than the all-in-one-cluster solution seen above.

Another workaround would be to force the clusters to be some given size. However, again it is unlikely that we can make any intelligent guesses about what sizes should be used in this kind of constraint. It seems more sensible to modify the objective function to not only favour clusterings that cut the smallest weight of edges, but also have reasonably similarly-sized clusters.

Figure 2.3: As the vertex v is well separated from the other points—and thus has very low affinity to all other points—a good Ratio Cut solution places v in a single cluster. However, v is clearly an outlier, distracting us from the more interesting problem of separating the grey vertices from the black. The Normalised Cut objective factors v's low total affinity into the objective function.

With this in mind, researchers [57] define the Ratio Cut problem (for k clusters), which balanced cluster size in a fairly straightforward way:

Problem: Ratio Cut

Given: A weighted graph $G = (V, E, w)$,
Find: A clustering $\mathcal{C}$ to minimise:

$$\text{Ratio Cut}(\mathcal{C}) = \sum_{i \in [k]} \frac{\text{cut}(C_i, \bar{C}_i)}{|C_i|} \,.$$

The Ratio Cut objective function certainly serves to achieve our goal of reasonably-sized clusters in our clustering, however there are some problems with it. Namely, vertices that are distant from most other vertices (outliers) and thus have small *degree* (total affinity over all incident edges) tend to distort the clustering, as they have equal influence as all other vertices. As they are outliers, it is sensible to try and minimise their impact, and focus on vertices which are more central. Figure 2.3 provides an example. For this reason, we can use a measure of cluster size which is a little more sensitive than purely the number of nodes. Remembering that:

$$\text{vol}(C_i) = \sum_{v \in C_i} \deg(v) \,,$$

Shi and Malik [96] define:

Problem: Normalised Cut

Given: A weighted graph $G = (V, E, w)$,
Find: A clustering $\mathcal{C}$ to minimise:

$$\text{Normalised Cut}(\mathcal{C}) = \sum_{i \in [k]} \frac{\text{cut}(C_i, \bar{C}_i)}{\text{vol}(C_i)} .$$

Previous Work

History of Spectral Clustering The Affinity Clustering problems as described have a standard solution technique, with wide application, known as Spectral Clustering—which we will describe in detail in Chapter 3—a technique to relax the combinatorial problem and to approximate it by solving a linear algebra problem.

As linear algebra problems are very well studied, and have some very mature and effective algorithms and software packages, Spectral Clustering is a comparatively quick algorithm. Additionally, the strong results that Spectral Clustering produces indicate that fundamentally it is a good relaxation, and the original problems it aims to solve are well motivated.

Spectral Clustering has been described many times, in many different communities, notably the work of Fiedler [45]; however, in the Computer Science community, Shi and Malik [96] were the first to use the spectral method to solve Normalised Cut. They clustered images into segments—contiguous region corresponding to a single object, or part of an object.

Meilă and Shi [81] show an interpretation of Spectral Clustering as a random walk (rather than a relaxed solution to Normalised Cut), with clusters corresponding to contiguous regions in which the walk remains. Another interpretation [85], connecting Spectral Clustering to perturbation theory is possible. These many justifications of the Spectral Clustering methodology are encouraging.

On the other hand, Nadler and Galun [84] provide some fundamental criticisms of the spectral approach, and of the local to global clustering scheme in general. They

demonstrate that certain types of problems, where clusters have different scales of density, cannot be solved by spectral approaches.

Applications of Spectral Clustering Spectral Clustering has been used to solve many real world clustering problems, in addition to the image segmentation applications mentioned above. Many applications have been in the bioinformatics area. Notably, Paccanaro et al. [87] show an application to clustering protein sequences, while Xing and Karp [108] show how normalised cuts can be used to cluster micro-array data, coincidentally with feature selection. This data captures genetic information about biological samples.

In a more mathematical application, Spielman and Teng [98] show that spectral techniques perform well on bounded-degree planar graphs and finite element meshes. These graphs are a superset of the majority of graphs found in real world applications.

Semi-Definite Programs As we will see in Chapter 3, an alternative relaxation for the Affinity Clustering problems is to semi-definite programs (SDPs). This approach has been investigated by a small group of researchers.

Xing and Jordan [107] found the first SDP extension to the spectral technique. Although compelling as a generalisation of Spectral Clustering, the SDP was not particularly well motivated as a relaxation of the Normalised Cut problem. Additionally it was complicated, and very expensive to run.

De Bie and Cristianini [31] improved this SDP with a much more motivated relaxation, similar to the famous SDP of Goemans and Williamson [54] for the Max-Cut problem. This algorithm achieved better results, and we will focus on this SDP further in Chapter 3. De Bie [30] gives a good survey of existing techniques to apply SDPs to machine learning.

2.2.4 Correlation Clustering Problems

In this section, we focus on problems that have to consider both *must-link* and *cannot-link* advice. So when clustering we have to consider both a Max-Cut and a Affinity Clustering problem at the same time.

Applications that lead to the above problem involve situations where pair-wise information is easier to discover than more global information. An early example was psychological experiments: two people might be friends, enemies or indifferent, and we want to form groups of friendly people (or at least, non-enemies). We can represent such a problem by a graph with edges representing feeling between two people, and a labelling of $+$ indicating friendship and $-$ indicating dislike, a signed graph of Section 2.1.

The Max-Cut problem that we have seen before corresponds to a situation where we ignore the information about friends, and concentrate on maximising the number of enemies that are separated. The Affinity Clustering problems correspond to concentrating on creating groups of friends—ignoring dislikes. The Correlation Clustering problems that we will define below ask us to concentrate on both.

Correlation Clustering asks us to consider both the $+$ and the $-$ edges. As usual, we can define both a maximisation and minimisation version. Correlation Clustering does not *require* that we need to specify the number of clusters—the $+$ edges will tend to lead us to bigger clusters, and the $-$ edges to smaller ones. The maximisation version is thus

Problem: Max-CC

Given: A signed graph $G = (V, E, l)$;
Find: A clustering $\mathcal{C}$ to maximise:

$$\text{Max-CC}(G, \mathcal{C}) = \big|(u, v) \in E^+ : \mathcal{C}(u) = \mathcal{C}(v)\big| + \big|(u, v) \in E^- : \mathcal{C}(u) \neq \mathcal{C}(v)\big|$$

Similarly, we have

Problem: Min-CC

Given: A signed graph $G = (V, E, l)$;
Find: A clustering $\mathcal{C}$ to minimise:

$$\text{Min-CC}(G, \mathcal{C}) = \big|(u, v) \in E^+ : \mathcal{C}(u) \neq \mathcal{C}(v)\big| + \big|(u, v) \in E^- : \mathcal{C}(u) = \mathcal{C}(v)\big|$$

The fact that we do not need to specify the number of clusters is quite an attractive aspect of the problem—most other clustering problems (such as Max-k-Cut) require it, and the choice is often quite arbitrary and unmotivated. However, when we do have a requirement on the number of clusters (see examples below), the Max-k-CC and Min-k-CC problems refer to restricted versions.

Again, a weighted version of these problems can be defined. Weighted Correlation Clustering operates on weighted complete graphs, where the weight function $w : E \to [0, 1]$ maps each edge to to 'how negative' it is.

A value of 1 indicates a $-$ edge, and a value of 0 indicates a $+$ edge. A value in between is less sure—to some extent it is a $+$ edge, to some degree it is a $-$ edge. Thus a value of $1/2$ indicates no preference as to whether that edge should be cut. The objective function (for Max-CC) is:

$$\text{Max-CC}(G, \mathcal{C}) = \sum_{\substack{e=(u,v)\in E \\ \mathcal{C}(u)=\mathcal{C}(v)}} (1 - w(e)) + \sum_{\substack{e=(u,v)\in E \\ \mathcal{C}(u)\neq\mathcal{C}(v)}} w(e)$$

As weighted Correlation Clustering operates on simple weighted graphs, we can make a connection to weighted Max-k-Cut and Min-k-Uncut.

$$\text{Max-}k\text{-CC}(G, \mathcal{C}) = \text{Max-}k\text{-Cut}(G, \mathcal{C}) + e_c(\mathcal{C}) - \text{Min-}k\text{-Uncut}(G, \mathcal{C})$$
$$\text{Min-}k\text{-CC}(G, \mathcal{C}) = \text{Min-}k\text{-Uncut}(G, \mathcal{C}) + e_u(\mathcal{C}) - \text{Max-}k\text{-Cut}(G, \mathcal{C})$$

This is a connection that we will exploit heavily in Chapter 5.

Previous Work

Unconstrained Correlation Clustering The Correlation Clustering problem was first investigated within the theory community by Bansal et al. [12]. They provided a reduction from Partition into Triangles to prove that the problem was NP-complete in the unweighted case. They also provide a PTAS for general 0/1-Max-CC, based on a randomised sampling procedure. They also construct a constant-factor approximation algorithm for 0/1-Min-CC, whilst proving the general weighted case is APX-hard.

Charikar et al. [21] showed that Min-CC is APX-hard even on 0/1 instances.

	MAX		MIN	
	unrestricted	fixed-k	unrestricted	fixed-k
k-Cut	N/A		N/A	
$\vdash$ 0/1		0.878 [54; 72]		
$\vdash$ dense		PTAS [6]		
$\vdash$ metric		PTAS [44]		PTAS [60; 43]
k-CC				
$\vdash$ 0/1	PTAS [12]	PTAS [53]	3 [12; 21; 2] APX-hard [21]	PTAS [53]
$\vdash$ weighted	0.7666 [21; 99] APX-hard [21]		APX-hard [33; 21]	
$\vdash$ metric		PTAS (∗)	2 [2]	PTAS (∗)
$\vdash$ consensus		PTAS (∗)	11/7 [2] APX-hard [18]	PTAS (∗)

Table 2.1: Summary of approximation results for k-CC and k-Cut type problems. The contributions made in Chapter 5 are indicated by (∗). Note that in most cases a problem is APX-hard if the number of output clusters is not restricted, but has a PTAS if there is a k-bound.

They also vastly improved the approximation factor provided by Bansal et al. (which was extremely large, whilst still remaining constant in n). Ailon et al. [2] further improved the approximation factor to 3, using a Quicksort-like algorithm, which we will discuss further in Section 2.3.4.

Charikar et al.'s algorithm was based on rounding a Quadratic Program, similar to a Semi-Definite Program, and also provided a constant factor approximation of 0.7664 for Max-CC on weighted graphs. This factor was improved slightly by Swamy [99] to 0.7666. Additionally, Demaine et al. [33], as well as Charikar et al., showed that the general Min-CC problem is APX-hard.

2-Correlation Clustering The Min-2-CC problem has been repeatedly re-discovered, and renamed, since it was first defined by Harary [58] in 1950. Harary introduced the signed graph, whilst considering the Min-2-CC problem. He also introduced the notion of *imbalance* in a signed graph, which corresponds to the Min-2-CC* cost of the graph, the minimum possible amount of violated advice. Harary considered the psychological interpretation of the problem: his aim was to

find two highly *cliquey* groups.

Apart from social psychology, the study of signed graphs has many other applications, notably in statistical mechanics, where it relates to energy configurations of the Ising model with no external field [82], and to the theory of graphs in Chemistry [102]. Solé and Zaslavsky [97] show a connection to coding theory: between signings of a graph and the cut-set code defined by that graph.

Dasgupta et al. [28] provide a particularly general and interesting application of the Min-2-CC problem, in the decomposition of large-scale biological systems. Here a + edge indicates experimental evidence that two components of the system are involved in the same biological processes; a − edge that they are not. The problem is then to separate the biological system into monotone subsystems—that is, subsets of the the system that are involved in the the same processes. Dasgupta et al. use their Min-2-CC algorithm as a step in a decomposition in to k clusters, in a hierarchical fashion.

k-Correlation Clustering Early results [95] demonstrated that k-CC is an NP-complete problem, both on complete graphs, and in general.

In Bansal et al. [12]'s paper on general Correlation Clustering, they also put forward the first approximation algorithm for Min-k-CC on *complete* graphs, achieving an approximation factor of 3. It provides the inspiration for a number of algorithms that we introduce in this thesis, and is defined:

Algorithm: Pick-a-Vertex

For each vertex $v \in V$, define a clustering $\mathcal{C}^v$, defined as

$$\mathcal{C}^v(u) = \begin{cases} +1 & \text{if } u \in N^+(v), \text{ or } u = v, \\ -1 & \text{if } u \in N^-(v)\,. \end{cases}$$

Return the clustering $\mathcal{C}^v$ that minimises Min-2-CC($\mathcal{C}^v$).

Pick-a-Vertex runs in $O(n^2)$ time, examining each edge twice.

Giotis and Guruswami [53] completed the picture—from a *theoretical* viewpoint—for k-CC on complete graphs by developing a PTAS (polynomial time approximation

scheme) for both the maximisation and minimisation versions of the problem. They first take a random sample of the vertices and then use each possible clustering of the sample as a basis for a clustering of the entire data set.

Recall that for Correlation Clustering on complete graphs, a PTAS exists for maximisation, but minimisation is APX-hard [21].

Giotis and Guruswami's scheme provides a $(1+\epsilon)$-factor approximation algorithm that runs in time $2^{O(1/\epsilon^3)}$. However, the constants involved are large enough that the smallest possible sample size is greater than 4000. In practice, checking every sample clustering is infeasible, so this algorithm remains beyond practical application. We will investigate variations of this algorithm that run in reasonable amounts of time in Chapter 4.

On *general* graphs, the problem is more difficult to solve. There is a direct relationship between Min-2-CC and Max-Cut problem: replace all + edges on the signed graph with a pair of − edges meeting at a new vertex. Dasgupta et al. [28] extend the Goemans Williamson SDP for Max-Cut to result to the maximisation version of the Min-2-CC problem, achieving the 0.878-approximation factor. It is not clear whether this algorithm is efficient in practice for the Min-2-CC problem—we will investigate further in Chapter 4.

Finally, Huffner et al. [59] use a fixed parameter algorithm, and some data reduction rules, to solve Min-2-CC exactly in greatly reduced time compared to a brute force algorithm. This is an example of the alternate approach to solving NP-complete optimisation problems—to accept super-polynomial running time, and to try and find (relatively) fast improvements on brute-force algorithms.

2.2.5 Affinity Clustering with Advice

In the previous sections, we discussed techniques that solve Affinity Clustering problems through relaxations, specifically the Spectral Clustering relaxation, and mentioned their wide application in the experimental clustering community. Such algorithms are very useful when we have a large corpus of data in which we have a good idea of the associations between points. However, it is possible that we also have some combinatorial information also, in the form of *advice*, similar to the problems we have seen previously.

In the Affinity Clustering with Advice problem then, we have two graphs defined over V, a *affinity graph*, which defines a Affinity Clustering problem, and a *advice graph*, which defines a Correlation Clustering problem. We aim to solve the first problem whilst respecting as much of the advice as possible.

Note that there are similar problems where the clustering is adapted to incorporate *constraints*. A constraint forces us to cluster vertices together or apart; on the other hand advice simply asks us to. A constraint problem thus implicitly trusts the source of the constraints; we consider applications where the advice is not necessarily trustworthy. For example, in biology, when experimentally clustering proteins (or genes etc), we will have a measure of the similarity of any pair of objects (the affinity graph). It is often practical to also test associations of individual pairs, which will tell us whether or not a pair should be clustered together (the advice graph). However, there is no guarantee that the advice we generate in this way will be correct.

Additionally, it is well known that human and biological 'experiments' are often subject to noise. If we have enough noisy advice, that advice will be inconsistent—that is, there is no way to cluster the data which agrees with all the advice. In this context, a problem—or an algorithm based on such a formulation—that strictly obeys the advice cannot function well, if at all.

Affinity Clustering with Advice differs from the other problems that we consider in this thesis, in that it does not have an explicit objective function. Below, we will see the approaches previous researchers have used to construct one; including a straightforward combination of objective functions, as well as using the advice to influence the affinities. In Chapter 3, we will provide our own construction.

Previous Work

The problem of clustering affinity data with constraints has been well studied in the Machine Learning community. The problem was first outlined by Wagstaff and Cardie [105], who extended the COBWEB algorithm of Fisher [47] to use constraints. Many other researchers have adapted existing clustering algorithms (usually designed to solved different objective functions) to incorporate constraints. Of note are algorithms that adapt (the popular) Lloyd's algorithm [77] for the

k-MEANS problem. Bilenko, Basu, Mooney [15] create a variant that involves incrementally changing the metric, simultaneously to the steps of LLOYD'S algorithm, to respect the constraints. This is a similar idea to the metric-changing algorithms described below.

Another LLOYD'S variant was developed by Davidson and Ravit [29]. They change the objective function used by that algorithm from the classic least squares k-MEANS objective, to one that punishes each failure to respect a constraint. They term this algorithm the CVQE algorithm. Pelleg and Ballas [90] provide a notable improvement to this algorithm that can cope with noisy constraint sets.

Other researchers have attempted to change the metric underlying a affinity clustering problem in an attempt to integrate advice. Xing, Ng, Jordan and Russell [109] show how to learn a Mahalanobis metric using Newton's method for a simple problem, and a gradient-descent approach for the general case. Bar-Hillel et al. [13] use principal component analysis to do the same. These techniques have the advantage of being agnostic to the clustering algorithm that is then used on the resulting metric. The boosting techniques of Liu, Jin, and Jain [76], which change a clustering using advice, can also have this advantage.

Many other algorithms have been developed to incorporate constraints and advice into different clustering formulations. However, as the underlying clustering problem that they attempt to solve is not an AFFINITY CLUSTERING problem, it is difficult to judge which would be ideal for AFFINITY CLUSTERING. This is a general difficulty with the wealth of clustering criteria.

There are some algorithms which attempt to incorporate constraints into SPECTRAL CLUSTERING (the usual solution to AFFINITY CLUSTERING problems). Of particular interest is the algorithm of Kamvar, Klein and Manning [63], called "spectral learning" which works via modifying the Laplacian matrix used in the SPECTRAL CLUSTERING algorithm. Essentially, they set the affinity between two vertices to be 1 if they are linked by a + edge, and 0 otherwise (with some relevant scaling). These changes seem arbitrary and unmotivated, as they ignore the affinity information between those vertices, as well a choosing a new affinity that depends on the distribution of other affinities in a problematic way.

To highlight this point, consider the following two scenarios. If most vertices are reasonably uniformly separated, then + edges will not have a very large effect,

as the strongest affinity is not significantly higher than any other affinity. On the other hand, if a single pair of vertices have much higher affinity than all others—perhaps the same data-point sampled twice with noise—then a + edge will force the end-points similarly close. But which situation happens depends entirely on the distribution of the affinity data, and is not controlled by the algorithm implementer. This is an unsatisfactory situation.

The Subspace Trick An interesting method to combine advice into an algorithm that uses relaxation—such as Spectral Clustering—is to transform the combinatorial advice into continuous constraints. In the linear algebra setting, which the Spectral Clustering algorithm operates in, this results in what is known as the 'subspace trick'.

It was first outlined by De Bie, Suykens and De Moor [32], and Yu and Shi [110]. We will extend the technique and provide further motivation in Chapter 3.

2.2.6 Consensus Clustering

The problems we have been discussing so far have involved taking local, pair-wise data, and turning it into a global organisation of the graph. Although there are plenty of applications in which such pair-wise data is available, an important source of such problems involves reducing problems with global data to pair-wise problems.

One example of such a problem is Consensus Clustering. This problem asks us to find a representative clustering for a input set of clusterings. Consensus Clustering can be used to combine the wealth of different clusterings that different clustering algorithms provide. This is of much help, as there are many formulations of clustering, and thus different algorithms can be aiming to maximise very different objective functions—it isn't always clear which should be used. So we can have a wealth of clusterings of a given data set, with no clear idea of which is 'right'. It makes sense to try and amalgamate these clusterings into one clustering which represents the *consensus* of the collection.

The connection between these Consensus Clustering and the pair-wise clustering problem we have defined above is due to the choice of measure of the quality of the consensus.

Problem: Consensus Clustering

Given: A set of clusterings $\mathbb{C}$ over a common set V;

Find: A clustering $\mathcal{C}$, to minimise

$$\textsc{Consensus Clustering}(\mathbb{C}, \mathcal{C}) = \frac{1}{|\mathbb{C}|} \sum_{\mathcal{C}' \in \mathbb{C}} \mathcal{M}(\mathcal{C}, \mathcal{C}')$$

where

$$\mathcal{M}(\mathcal{C}, \mathcal{C}') = \big|u, v \in V : \mathcal{C}(u) = \mathcal{C}(v) \text{ and } \mathcal{C}'(u) \neq \mathcal{C}'(v) \text{ or } \mathcal{C}(u) \neq \mathcal{C}(v) \text{ and } \mathcal{C}'(u) = \mathcal{C}'(v)\big|$$

The measure $\mathcal{M}$ (the Mirkin [83] distance between two rankings, also known as the Rand index) leads to a connection between Consensus Clustering and Correlation Clustering because of the pair-wise nature of the distance function—two clusterings are measured to be different by the number of pairs of points that are clustered differently, relative to each other. Certainly other distance measures are possible, but this is a very natural formulation, and is commonly used.

To see exactly how this plays out, consider the following reduction, from Consensus Clustering to weighted Correlation Clustering.

Reduction 1. *Let a set of clusterings $\mathbb{C}$ over V be an instance of* Consensus Clustering. *Construct an instance of* Correlation Clustering *on the following weighted graph $G = (V, E, w)$, where $E = \{(u, v) \mid u, v \in V\}$ (so G is complete), and, for $u, v \in V$,*

$$w(u, v) = \frac{\big|\mathcal{C}' \in \mathbb{C} : \mathcal{C}'(u) \neq \mathcal{C}'(v)\big|}{|\mathbb{C}|}$$

.

Proposition 1. *Reduction 1 is an approximation preserving reduction.*

Proof. Let $\mathcal{C}$ be any solution to the Correlation Clustering problem on $G = (V, E, w)$. Consider the cost of $\mathcal{C}$ as a solution of the original Consensus Clus-

TERING problem.

$$\begin{aligned}\textsc{Consensus Clustering}(\mathbb{C}, \mathcal{C}) &= \frac{1}{|\mathbb{C}|} \sum_{\mathcal{C}' \in \mathbb{C}} \mathcal{K}(\mathcal{C}, \mathcal{C}') \\ &= \sum_{\substack{u,v \in V \\ \mathcal{C}(u) \neq \mathcal{C}(v)}} w(u,v) + \sum_{\substack{u,v \in V \\ \mathcal{C}(u) = \mathcal{C}(v)}} (1 - w(u,v)) \\ &= \textsc{Correlation Clustering}(G, \mathcal{C})\end{aligned}$$

So, as clustering will have the same cost in both problems, they share optima—and a α-approximate solution to one will be an α-approximate solution to the other. □

In fact, the instances of CORRELATION CLUSTERING that are generated by Reduction 1 have an additional property; the triangle inequality:

Proposition 2. *Let $G = (V, E, w)$ be an instance of* CORRELATION CLUSTERING *generated by Reduction 1. Then, for all $u, v, x \in V$,*

$$w(u,v) \leq w(u,x) + w(x,v)$$

Proof. Consider any clustering $\mathcal{C}' \in \mathbb{C}$. Then, if $\mathcal{C}'(u) \neq \mathcal{C}'(v)$, then either $\mathcal{C}'(u) \neq \mathcal{C}'(x)$ or $\mathcal{C}'(x) \neq \mathcal{C}'(v)$ (or possibly both if there are at least three clusters). So if $\mathcal{C}'$ contributes $\frac{1}{|\mathbb{C}|}$ to $w(u,v)$, it must contribute at least the same amount to the sum $w(u,x) + w(x,v)$. Running over all $\mathcal{C}' \in \mathbb{C}$ completes the proof. □

For this reason, the instances of CORRELATION CLUSTERING that are generated by reduction from CONSENSUS CLUSTERING are a subset of all instances of CORRELATION CLUSTERING with probability constraints and the triangle inequality. We refer to such instances as *metric*-CORRELATION CLUSTERING problems (this is the same definition as for metric-MAX-CUT problems). Note that CONSENSUS CLUSTERING can also be defined as a maximisation problem—the same reduction and properties hold.

Previous Work

Ailon et al.'s QUICKSORT-like algorithm, which we previously discussed in reference to CORRELATION CLUSTERING, is also 2-approximation for metric-MIN-CC and, further, and a 11/7-approximation for instances of metric-MIN-CC resulting from CONSENSUS CLUSTERING. However, Bonizzoni et al. [18] showed that a significant improvement to this result (a PTAS for the problem) is unlikely by demonstrating that CONSENSUS CLUSTERING problem is APX-hard, even with just three input clusterings. Remembering that Giotis and Guruswami were able to overcome the APX-hardness of 0/1-MIN-CC by restricting the problem to MIN-k-CC, this leads one to speculate whether the CONSENSUS CLUSTERING problem, restricted to k output clusters, is also APX-hard or has a PTAS. We answer this question in Chapter 5.

Consensus clustering can be used to form one representative 'meta'-clustering when a series of clusterings of a common dataset is known. Filkov and Skiena [46] show an application to clustering micro-array data as a bioinformatics application. Gionis et al. [52] perform an extensive study of the CONSENSUS CLUSTERING problem, comparing many of the existing algorithms. Bertolacci and Wirth [17] follow this up with a experimental study comparing many of the existing theorically-justified algorithms with some simpler heuristics.

2.3 Ranking Problems

The second class of problems that we consider are ranking problems on directed graphs. These problems ask us to rank, or order, the vertices of the digraph from left to right, whilst trying to follow advice given by the arcs. An arc $u \to v$ on the digraph represents advice that u should be ranked before v. So our problem is to take the local information encoded by the arcs, and construct a global ordering over all the vertices.

If the digraph is acyclic—it is a *Directed Acyclic Graph* (DAG)—we can form a ranking over the entirety of V by taking a TOPOLOGICAL SORT of G. A TOPOLOGICAL SORT of a DAG G is any ranking of V that agrees with every arc in G. A simple depth-first search algorithm for TOPOLOGICAL SORT, by Tarjan [101] can

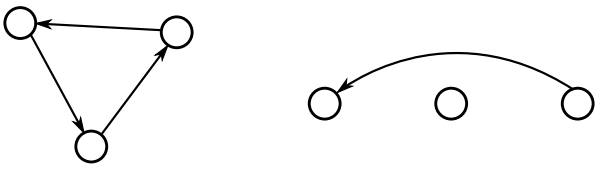


Figure 2.4: A 'bad triangle' for MIN-FAS. There is no ordering of the three vertices that respects all three arcs. The best possible ordering has a single back-arc, as shown. Note that on the right-hand side, only the back-arc is pictured; there are two arcs that point to the right that are not pictured.

achieve this in time $O(n + m)$. This is quite a satisfying situation as we have perfectly turned the local information encoded in the graph into a global property of the digraph.

However, if the digraph is not a DAG—there is a cycle in the graph—then we cannot find any order over the vertices in which every arc in G is respected. For example, suppose we have the arcs $u \to v, v \to x, x \to u \in A$. Then there is no ordering of u, v, x that agrees with all three arcs. Figure 2.4 demonstrates this situation. Note that in this picture, as with all *tournament* diagrams, when the vertices are ordered (as they are on the right) only the backwards (leftwards) pointing arcs are drawn. It is assumed that any pair of vertices that have no arc between them have a rightward pointing arc.

This is analogous to the triangle of edges situation that is difficult for MAX-CUT (see Section 2.2.1). The challenge is then to find a ranking of V which is as consistent as possible with G. So in fact the cycles in G encode the difficulty in ranking G.

2.3.1 MIN FEEDBACK ARC SET

A simple example is a sports competition. Suppose we have a competition where some teams play each other and we need to form a final ranking of the teams. Then the games that have been played tell us something about how the final ranking should look—if team A beat team B that is an indication that A should be ranked above B in the final ordering. So each game forms an edge in our 'competition' graph. If the competition was round-robin (each team played each other once) then the graph that results is a tournament graph.

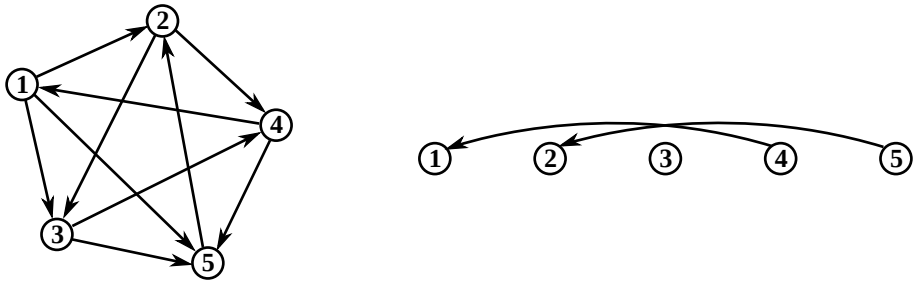


Figure 2.5: A demonstration of the Min Feedback Arc Set objective function. An optimal arrangement of the tournament on the left is displayed on the right. This ranking has cost 2, indicated by the two back-arcs.

How do we represent this ranking (or sporting ladder)? In this thesis, we take a functional approach. So

Definition 10. *A ranking π is a bijection from the vertices V to $[n] = \{1, 2, 3, \ldots, n\}$ where $n = |V|$. If $\pi(u) > \pi(v)$ we say u is ranked after (to the right of) v.*

A ranking can also be viewed in a more traditional way as a permutation of V. Then, without loss of generality, we abuse notation and write

$$\pi = (v_1, v_2, \ldots, v_n) \text{ where } \pi(v_i) = i$$

The natural problem to consider in this context is to find a ranking which agrees as much as possible with the digraph. Here the input information is a directed graph, where an arc $u \to v \in A$ indicates that π should rank u before v. So we have an obvious minimisation problem:

Problem: Min Feedback Arc Set (Min-FAS)

Given: A directed graph $G = (V, A)$;
Find: A ranking π to minimise:

$$\text{Min-FAS}(G, \pi) = |u \to v \in A : \pi(u) > \pi(v)|$$

The Min-FAS objective function thus counts the number of arcs that point from right to left; such arcs are sometimes called 'back-arcs'. A demonstration of the Min-FAS objective function is given in Figure 2.5.

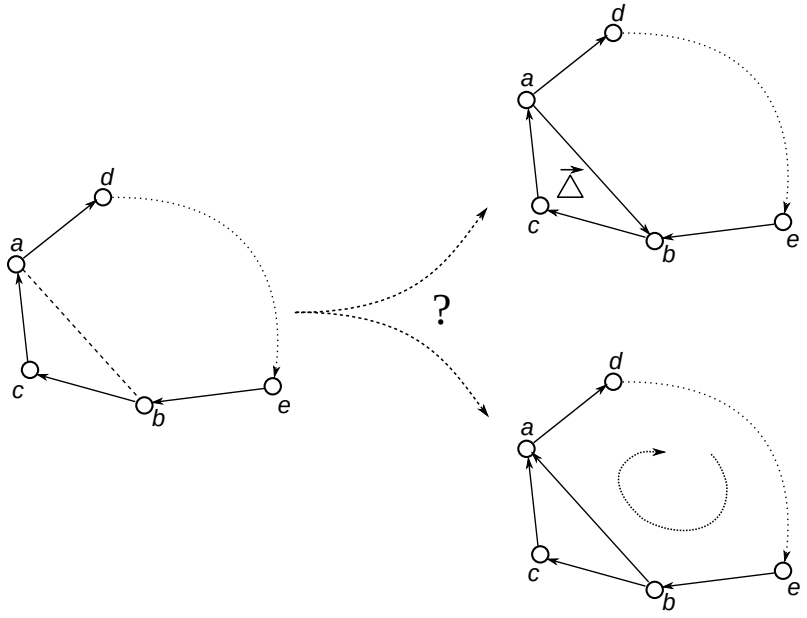


Figure 2.6: When dealing with a tournament, a directed cycle must contain a directed triangle. If $a \to b \in A$, then $a \to b \to c \to a$ is a directed triangle ($\vec{\triangle}$). If $b \to a \in A$, $a \to d \to \cdots \to e \to b \to a$ is a smaller cycle, and we can recurse. As the the graph is a tournament, one of the cases must hold.

When a tournament is laid out in some order, to form a cycle there must be an arc pointing left, so is easy to see that after removing all the back-arcs from G, what remains will be be acyclic.

For this reason, the set of back-arcs is known as a *feedback arc set*—as removing this arc set removes all feedback (cycles) from the graph. Min-FAS on tournaments is particularly interesting. One reason is that each cycle on a tournament must contain a directed triangle (see Figure 2.6). This gives special structure to the problem which we will exploit later.

Of course, Min-FAS has a complementary maximisation problem: this corresponds to counting the number of arcs that are not deleted—this set of arcs will form an acyclic subgraph of G. So the problem is:

Problem: Max Acyclic Subgraph

Given: a directed graph $G = (V, A)$
Find: a ranking π to maximise:

$$\text{Max Acyclic Subgraph}(G, \pi) = |u \to v \in A : \pi(u) < \pi(v)|$$

2.3.2 Weighted Problems

Correspondingly with Correlation Clustering, we consider a specific type of weighted graph for the Min-FAS problem. We have as input a weighted tournament, (V, A, w). The weight $w(u \to v)$ is the extend to which u should be placed before v—we incur a cost of $w(u \to v)$ if $\pi(u) > \pi(v)$ and a cost $1 - w(u \to v)$ otherwise.

The above formulation is a special case of a more general problem, known as the Linear Ordering [22] problem. In this problem, each pair of vertices has two arcs, one for each direction. Each arc has its own weight. Thus the need for the graph to be directed is no longer relevant. In fact, since the weights are really the only important quantities, we can use a matrix of the arc weights as a problem instance. So we have

Problem: Linear Ordering

Given: an $n \times n$ matrix $\mathbf{M}$,
Find: a ranking π to minimise:

$$\text{Linear Ordering}(\mathbf{M}, \pi) = \sum_{i,j \in [n] \,:\, \pi(i) < \pi(i)} m_{ij}$$

The Linear Ordering ordering problem is similar enough to Min-FAS that we can often use algorithms designed for Linear Ordering for Min-FAS. Otherwise, we will not consider the problem further.

2.3.3 Rank Aggregation

Min-FAS is also related to a fundamental aggregation problem. This can be shown, corresponding to Consensus Clustering, to be a specification of weighted Min-FAS with the triangle inequality. The Rank Aggregation problem has many applications, including that of *Metasearch* [9]—combining the output of multiple search engines into a single, hopefully better, search result. The definition of Rank Aggregation is:

Problem: Rank Aggregation

Given: A set of rankings Π over a common set V;
Find: A ranking π, to minimise

$$\text{Rank Aggregation}(\Pi, \pi) = \frac{1}{|\Pi|} \sum_{\pi' \in \Pi} \mathcal{K}(\pi, \pi')$$

where $\mathcal{K}$ is the Kemeny distance [67] (or Kendall, or inversion distance), a measure of the number of pairs of vertices that are ordered differently:

$$\mathcal{K}(\pi, \pi') = \left| u, v \in V : \pi(u) < \pi(v) \text{ and } \pi'(u) > \pi'(v) \right|$$

2.3.4 Previous Work

Applications of Min-FAS. The Min Feedback Arc Set problem was originally motivated by problems in circuit design [62], although it has found application in many areas. In the general case it can be used to model constrained scheduling problems, such as are common in compiler design; it has applications in computational chemistry [73; 88], and graph drawing [40; 34].

Other applications, especially for the restriction to tournaments, come from the connection to Rank Aggregation. Dwork et al. [38] first outlined this connection, and motivated it as a method for aggregating data from search engines. There is a significant body of work studying this problem, which is known as MetaSearch [9].

Some authors consider the Rank Aggregation problem in isolation. One perspective is to treat the input rankings as a samples of some unknown, 'true'

ranking under some noise model. Given such a formulation, Meilă et al. [80] develop a maximum-likelihood estimator which will find the ranking which has the highest probability of being that 'true' ranking under an exponential noise model. Of course in many applications, there is no 'true' ranking, as ranking is usually a very subjective problem. However, Conitzer and Sandholm [24] add extra interest to the area by pointing out many commonly-used voting rules (which are of course simply Consensus Clustering algorithms) can be paired with particular noise models for which they are the maximum-likelihood estimators.

Heuristics for Min-FAS. The problem of Min-FAS is familiar to organisers of sporting leagues—often each team plays each other once, and a ladder must be constructed which ranks the teams as fairly as possible. In this context, it is natural to ask to minimise the number of upsets as a measure of 'fairness'. The standard heuristic for this problem is—unsurprisingly—a simple one. We rank each team by the number of wins it has achieved. In the graph version of the problem, where a match is represented by an arc pointing from the loser to the winner, this is simply the indegree. This means that the best teams (vertices) will be placed on the right hand side when we order by indegree. The indegree has traditionally been given another name, after Kendall [68], the first to propose this algorithm to solve Min-FAS. The *Kendall score* of a vertex v is simply v's indegree—that is the number of other vertices x, such that we have a statement of the form 'x should be ranked before v'.

The one complication to ordering by Kendall score is in resolving the issue of ties—what happens if two teams have the same number of wins? Sporting leagues tend to use domain-specific solutions to the problem (e.g. goal difference). Ali et al. [3] and Cook et al. [25] propose a more general solution: for a set of vertices $S \subseteq V$ of the same Kendall score, break the tie by performing the same process on $G|_S$, the subgraph induced by S. We refer to this algorithm as the Iterated Kendall algorithm, and it can be implemented in $O(n^2)$ time:

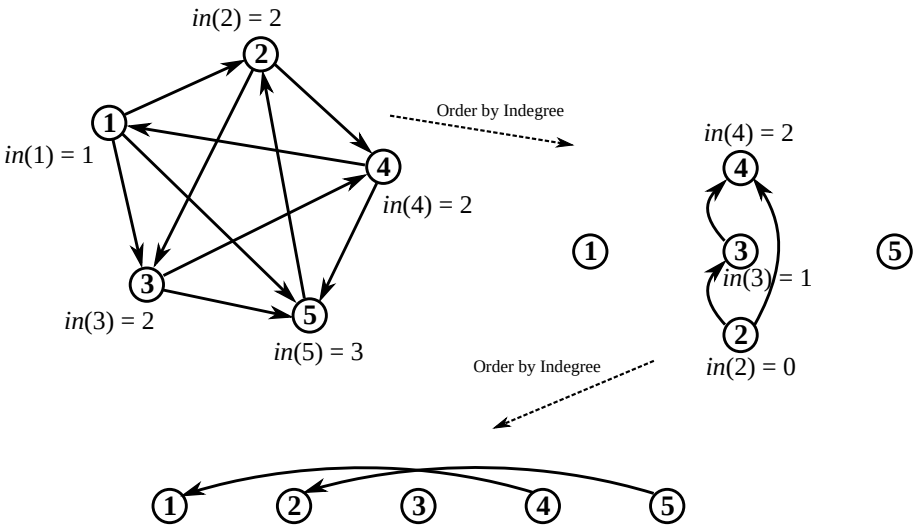


Figure 2.7: An example of Iterated Kendall in action. The set $\{2, 3, 4\}$ has equal indegree, so the algorithm iterates on that subset.

Algorithm: Iterated Kendall

1. Calculate the Kendall score of all vertices.
2. If all vertices have the same Kendall score, return an arbitrary ranking.
3. Let π_0 be a ranking such that $\pi_0(v)$ be the number of vertices with lower Kendall score than v.
4. For each subset $S \subseteq V$ of equal Kendall score (including singleton subsets), let $\pi_1 = \text{Iterated Kendall}(G|_S)$. For $v \in S$, let $\pi(v) = \pi_0(v) + \pi_1(v)$.
5. Return π.

An example of the Iterated Kendall algorithm in action is displayed in Figure 2.7.

Eades et al. [39] created an algorithm that is in fact quite similar to Iterated Kendall, possibly inspired by selection sort. Again we choose the vertex v of lowest Kendall score to place on the left-hand side of the ranking. The difference is that we then re-compute Kendall scores for the remaining vertices by considering the subgraph remaining after removing v. The algorithm is still $O(n^2)$:

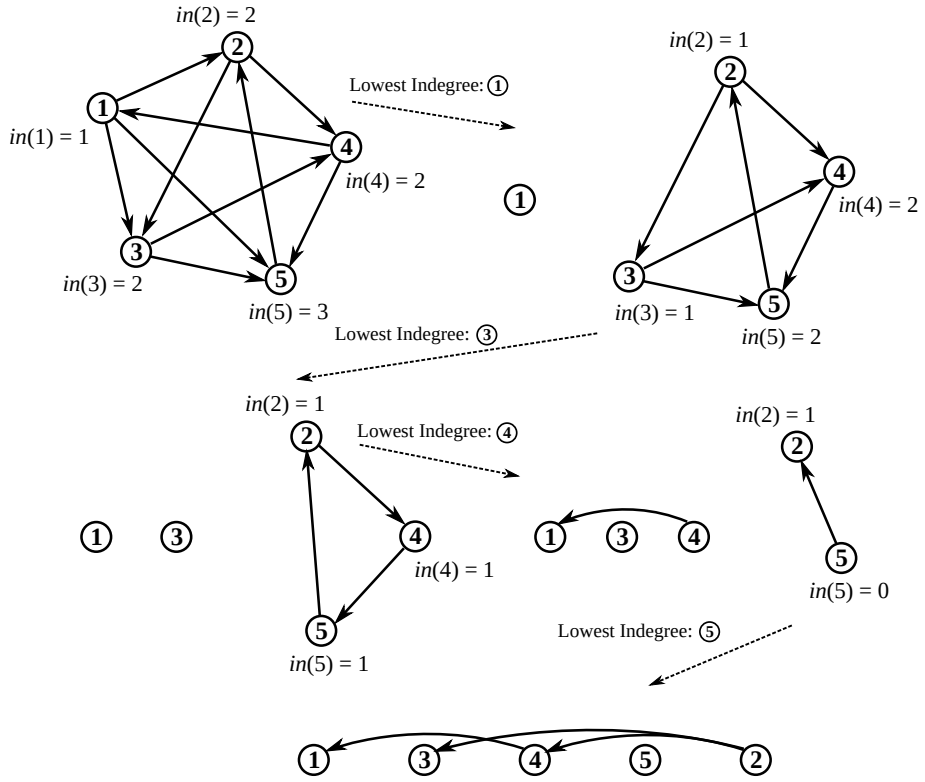


Figure 2.8: An example of EADES in action.

Algorithm: EADES

1. Choose the vertex m of minimal indegree.
2. Let $\pi_0 = \text{EADES}(G|_{V\setminus\{m\}})$.
3. Return
$$\pi(v) = \begin{cases} 1 & \text{if } v = m, \\ \pi_0(v) + 1 & \text{otherwise.} \end{cases}$$

An example of the EADES algorithm in action is displayed in Figure 2.8.

Sorting algorithms Figure 2.8 makes it clear how the EADES algorithm can be seen as an adaption of SELECTIONSORT. In fact, many sorting algorithms can be adapted in a straightforward way to apply to the MIN-FAS problem, which Bar-Noy and Naor [14] were the first to discuss. This is due to the similarity in structure between the MIN-FAS problem and the classic problem of sorting.

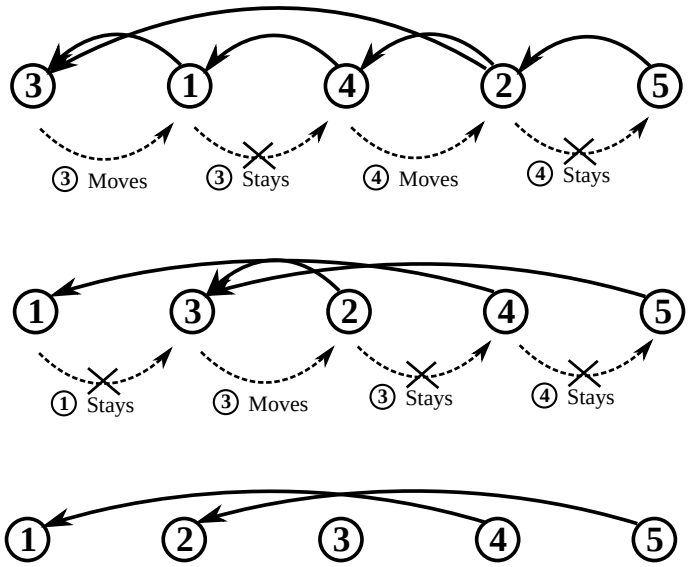


Figure 2.9: An example of BUBBLESORT in action.

Unlike traditional sorting problems, in which we assume there is a total order on the data, in FAS we have no such ordering. In fact the very difficulty in FAS is the lack of transitivity, which sorting algorithms are designed to exploit. Nevertheless, sorting algorithms provide schemes for deciding which of the advice to *believe*—we can use them to decide in which order to respect the arcs.

So we can define a general strategy for FAS based on some sorting algorithm $\mathcal{S}$; run $\mathcal{S}$ over the vertices of G, using as the *comparison* function "$u < v$ if and only if $u \to v$". On weighted instances this becomes "$u < v$ with probability $w(u \to v)$", leading to a randomised algorithm.

For instance, using this strategy, BUBBLESORT is defined as follows (of course running in $O(n^2)$):

Algorithm: BUBBLESORT

1. Begin with a random ordering π.
2. Consider each $j \in [n]$. If there is an arc $u \to v$, where $\pi(u) = j$ and $\pi(v) = j - 1$, then swap u and v.
3. If π changed in step 2, repeat.

An example of the BUBBLESORT algorithm in action is displayed in Figure 2.9. Note

that every step of the BUBBLESORT algorithm improves the cost of the ranking; in fact it is a variant of the SWAPS local-search heuristic we use in Chapter 4. For this reason, the algorithm must terminate.

Cook et al. [25] develop an algorithm designed to ensure that a Hamiltonian path exists along the ranking of the vertices; any sensible algorithm should achieve this. The method they use to achieve this is in effect a BUBBLESORT as described above.

Chanas and Kobylanski [19] apply an insertion technique to the LINEAR ORDERING problem that is more involved than the usual INSERTIONSORT. As a subroutine, they use what is in effect one run of INSERTIONSORT, which they name *SORT*. It is defined as follows:

Algorithm: SORT

Given an ordering π over a digraph G,

1. For $i \in [n]$, let v be the vertex such that $\pi(v) = i$, do the following:
2. For each $j \in [i]$, let $\pi_{i,j}$ be the ordering defined by moving v to position j, whilst leaving the remaining vertices in the same order.
3. Update π to be the $\pi_{i,j}$ that minimises MIN-FAS$(G, \pi_{i,j})$.
4. Next i.

Obviously this sort step runs in time $O(n^2)$. An example of the SORT step in action is displayed in Figure 2.10.

Since executing SORT cannot increase the number of back-arcs, the authors first propose an algorithm SORT* which repeatedly applies SORT until there is no improvement in the number of back-arcs. They also show that the composition of two steps SORT $\circ$ REVERSE (where REVERSE simply reverses the order of the nodes) cannot increase the number of back-arcs. This is because in effect SORT $\circ$ REVERSE is the same procedure operating from the right hand side. The CHANAS algorithm is therefore (SORT* $\circ$ REVERSE)*, which the authors have demonstrated outperforms SORT* alone on general graphs. The worst-case running time is then $O(n^4)$ as the initial cost is $O(n^2)$, however the running time tends to be much better as each step tends to greatly improve the cost.

Saab [94] presents an algorithm using a divide-and-conquer approach. The idea

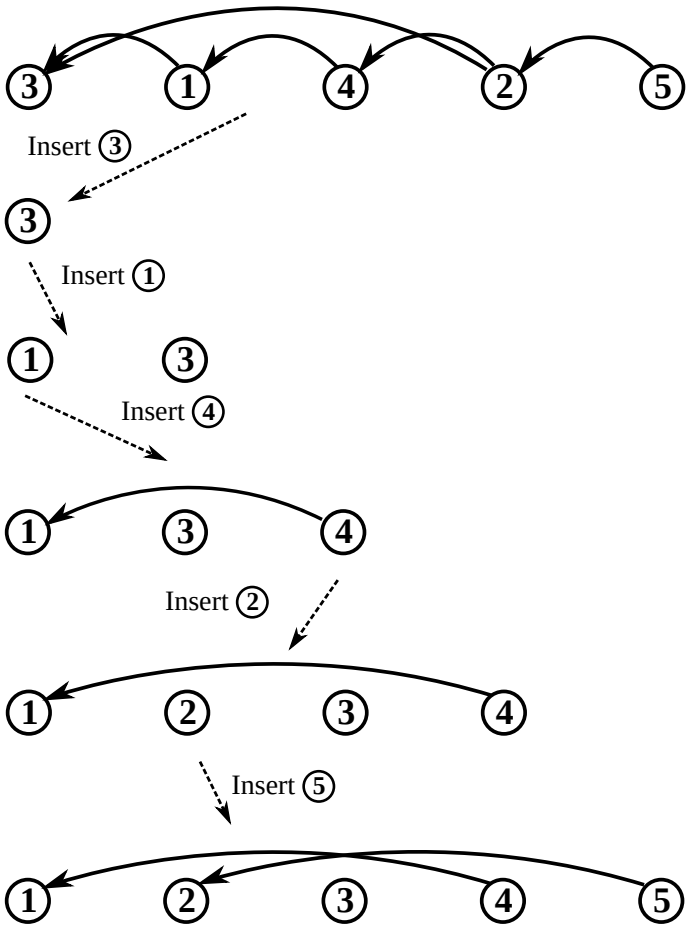


Figure 2.10: An example of a single step of SORT. Note that SORT $\circ$ REVERSE is simply the same process, beginning from the right-hand side (i.e. inserting nodes in the order $5, 2, 4, 1, 3$).

is to split the input into two halves, minimising the number of back-arcs between the halves, and then recurse on each half. However this minimisation task is a difficult one: it is a directed version of Min Bisection—the Min-Cut-style problem that asks us to cluster the vertices into two equal sized clusters—known to be APX-hard [72], and solving it would solve Min-FAS. So this algorithm is simply deferring the Min-FAS problem to a Min Bisection problem.

Theoretical Results When considered on general graphs, the Min-FAS problem is one of Karp's [66] original NP-complete problems. Dwork et al. [38] proved the Rank Aggregation problem is NP-complete, even with as few as 4 input rankings. Min-FAS restricted to unweighted tournaments was also conjectured [11] to be NP-complete, although the proof only came recently, first with randomised reductions [2]. Deterministic reductions were found by two groups of researchers independently [20; 4]. A NP-completeness proof for the case of bipartite tournaments (a directed complete bipartite graph) was also found by Gou et al. [56].

Given these results, it is natural to ask for an approximation algorithm which runs in polynomial time, yet is guaranteed to differ in cost from the optimal solution by a small factor.

In the general case, the best known algorithm is a $\log(n) \log\log(n)$-approximation algorithm by Even et al. [41]. In the bipartite case, Dom et al. [36] find fixed parameter algorithms for both Min-FAS and the closely related Feedback Vertex Set problem.

Recently, there has been a flurry of activity in the approximation algorithms community focused on the Min-FAS problem for tournaments. The first constant factor approximation algorithm for Min-FAS, designed primarily for tournaments, was the pivoting algorithm of Ailon et al. [2]. Intuitively similar to Quicksort, it runs in worst case $O(n^2)$ time, although it uses on average $O(n \log n)$ comparisons, for an expected 3-approximation on tournaments.

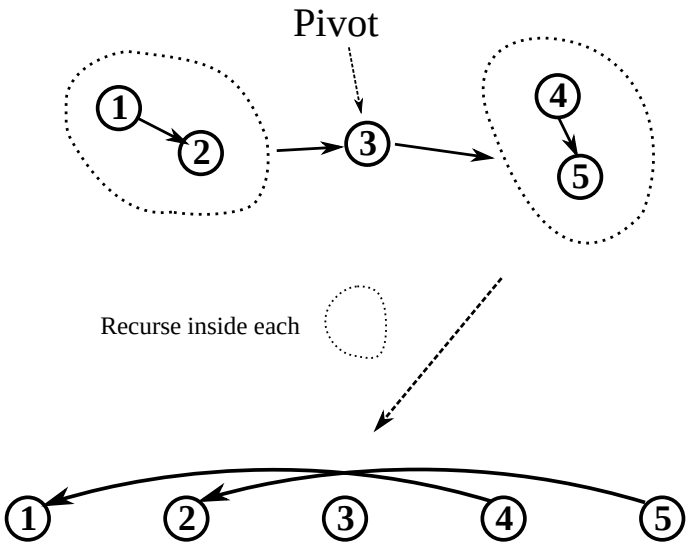


Figure 2.11: An example of QUICKSORT in action.

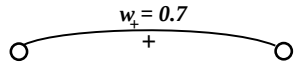
Algorithm: QUICKSORT

> Choose a pivot $p \in V$, uniformly at random. Let $L \subseteq V$ be all vertices v such that $v \to p$, and let $R = V \setminus (L \cup \{v\})$. Also, let π_L be the ordering of L obtained by QUICKSORT, and π_R be the analogous ordering of R. Output (π_L, v, π_R), the ordering resulting from placing vertices in L on the left (ordered by π_L), etc.

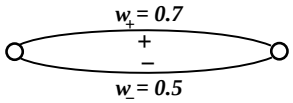
An example of the QUICKSORT algorithm in action is displayed in Figure 2.11. The QUICKSORT algorithm is also a 11/7 approximation for instances of MIN-FAS that are generated from RANK AGGREGATION problems.

Coppersmith et al. [26] showed that ordering the nodes by their Kendall score is a 5-approximator on tournament instances. This of course includes the ITERATED KENDALL algorithm. Interestingly, in terms of approximation guarantees, the method chosen to resolve ties is irrelevant. So an arbitrary choice is as good as the full recursive nature of the ITERATED KENDALL algorithm.

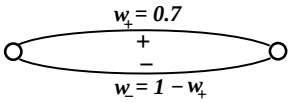
Van Zuylen et al. [103] find a *deterministic* 3-approximation for a specific case of MIN-FAS: a slight variation that aims to expand an existing ranking to a full ranking over all the vertices of the tournament. Their algorithm is a 2-approximation when that problem is metric (which includes RANK AGGREGATION as a special case).



(a) Simple weighted graph. Each pair of vertices can be + linked or − linked, with a single weight.



(b) General weighted graph. Each pair of vertices is both + linked and − linked, with any weight on both links.



(c) Weighted graph with probability constraints. Each pair of vertices is both + linked and − linked, but the weights must add to 1.

Figure 2.12: The three possible types of weighted graph, for the Correlation Clustering problem. The Min Feedback Arc Set problem has a similar set of three possibilities.

Kenyon-Mathieu and Schudy [69] completed the approximation picture for Min-FAS on tournaments with a PTAS (polynomial-time approximation scheme). This scheme comprises a local-search heuristic and the PTASes of Arora et al. [5] and Frieze and Kaplan [50] for dense instances of the Max Acyclic Subgraph problem.

2.4 Weighted Problems

Our description of weighted versions of the Min-FAS and Min-CC problems may seem at odds with standard formulations. Our formulation was a generalisation of the complete unweighted version of those problems. For example, in the complete unweighted version of Min-CC, every pair of vertices $u, v \in V$ has an edge $e = (u, v)$, with a label $l(e) = \pm 1$. If we were to form a clustering that agreed with that label, we would incur no cost; disagreeing with the label would incur cost 1.

In our weighted version, the edge had an additional *weight*, $w(e) \in [0, 1]$. This weight represents the cost of violating e; additionally, the cost of *not violating* e was defined to be $1-w(e)$. This generalises the 0/1 case, in which $w(e)$ was automatically 1. Our weighted version of the Min-FAS problem is similar.

There exist other weight-based generalisations of the original unweighted problems. There are two other generalisations that we will describe here, and we demonstrate Figure 2.12.

The first, and simplest generalisation of an unweighted problem is simply to incur

cost $w(e)$ if e is violated, and cost 0 otherwise. This is a very natural generalisation.

The second version is the most general, and contains both our formulation and the previous example as special cases. In this general version, each edge has two weights, representing the extent to which both possible restrictions apply. So in the case of Min-CC, an pair $u, v \in V$ has a positive weight $w^+(u, v)$ and a negative weight $w^-(u, v)$. If we separate u from v, we incur the cost $w^+(u, v)$; otherwise we incur the cost $w^-(u, v)$. In the Min-FAS case, we have two costs representing the two possible orderings of u and v; in actuality, this is the Linear Ordering problem, as we described it in Section 2.3.2.

Instances of this general weighted problem where one or both of the quantities $w^+(u, v)$ and $w^-(u, v)$ must be zero for every pair $u, v \in V$ results in the simple weighted case as described above. Instances where $w^+(u, v) + w^-(u, v) = 1$ for all $u, v \in V$ are said to have *probability constraints*. This is exactly our formulation of weighted graphs. Notice that in this case there is no need to record the negative weight function w^-, as it can be calculated from w^+, so we can specify the problem with a simple weighted graph. Similarly, we can use a weighted directed graph, with at most one arc between any pair $u, v \in V$, for weighted Min-FAS with probability constraints.

The reason that we restrict our attention to instances with probability constraints in this thesis is due to the fact that the weighted instances that we are interested in—namely those that result from the reductions from Consensus Clustering and Rank Aggregation—are guaranteed to have this quality.

Chapter 3

Relaxation

The optimisation problems that we have been concerned with have been exclusively *combinatorial*—problems in which discrete choices must be made. We can never say 'these two nodes should probably be in the same cluster'—we have to say definitively *yes* or *no*. However, there is a entire field of optimisation in which the answers which are sought are not so discrete. In *continuous* optimisation, where the solution space is continuous, there is a far larger cardinality of solution space. Perhaps surprisingly, this is often an advantage when it comes to designing optimisation algorithms.

The combinatorial optimisation problems we have seen so far have tended to either be trivial to solve, or NP-complete. On the other hand, there exist powerful algorithms which solve large classes of continuous optimisation problems in polynomial time. For example, linear programming (which we will discuss presently) can describe a vast number of continuous optimisation problems—a practical solution has existed since 1947 [27], and a polynomial time solution since 1979 [70].

For this reason, it is not surprising that one powerful technique in solving combinatorial problems involves 'turning' them into continuous problems. We do this by *relaxing* the discreteness of the solution—for example, rather than requiring a solution that definitively states which cluster each point should be in, we allow solutions where points are 'between clusters' to some degree, and solve the resulting continuous optimisation problem.

Of course, when we have solved the continuous problem, except for a few special problems, it is unlikely that the solution we get out fits the original combinatorial

problem. We will then need to find a combinatorial solution that is 'near' (what exactly is near is a very important part of the process) to the continuous solution. We call this part of the process *rounding* the continuous solution. Often we can show that this rounding will not change the cost of the solution very much, and thus we can find very good solutions in this way.

In this chapter we will outline the basics of relaxation and show how these techniques can be used to find a good solution to the general Affinity Clustering problems. We will outline two basic strategies: a Semi-Definite Programming (SDP) strategy, based on the work of De Bie and Cristianini [31]; and the more famous linear algebra strategy known as Spectral Clustering.

We will then expand both of those techniques to find a solution to the related problem of Affinity Clustering with Advice. We will do this by transforming the advice that we are given (the Correlation Clustering part of the problem) into sensible constraints on the continuous solution space.

3.1 Linear Programming

Any continuous optimisation problem consists of a set of real-valued variables (the solution space), some constraints on those variables, and some function that produces an objective value based on those variables. A linear program (LP) is a specific type of problem where the constraints and the objective are *linear* functions of the variables.

If we represent the variables by a vector $\mathbf{x} \in \mathbb{R}^n$, we can write every LP in the following form:

P1. *Linear Programming*

$$\min \quad \mathbf{c}^T\mathbf{x}$$

$$\text{s.t.} \quad \mathbf{A}\mathbf{x} \geq \mathbf{b} \qquad (3.1)$$

$$\mathbf{x} \geq \mathbf{0} \qquad (3.2)$$

Here, $\mathbf{A}$ is a $m \times n$ matrix (where m is the number of constraints), $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. The constraint (3.2) is purely a convention—if we allow negatively valued

variables, we can represent them with two non-negatively valued variables.

Linear programs can be solved in polynomial time, although the most popular algorithms, based on the SIMPLEX algorithm of Danzig [27], are not, in fact, guaranteed to run in polynomial time. However, ELLIPSOID [70] and INTERIOR POINT [65] algorithms exist that solve linear programs exactly in polynomial time. Note that the theoretically fastest known algorithms run in time $O(n^3 l)$, where n is the number of variables in the program, and l is a measure of the problem size (essentially proportional to the number of constraints).

The combinatorial problems which best relax to LPs are the Integer Programs (IPs). These are simply LPs with the added restriction that each x_i takes on integral values. Many combinatorial optimisation problems can be framed in such a manner. As an example, consider the problem of MIN SET COVER.

Problem: MIN SET COVER

GIVEN: a set V of n elements, a collection $\mathcal{S} = (S_1, \ldots, S_k)$ of subsets of V, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$,
FIND: a sub-collection $\mathcal{T} \subseteq \mathcal{S}$, such that $\mathcal{T}$ covers V, to minimise

$$\sum_{T \in \mathcal{T}} c(T)\,.$$

Set cover can be represented by the following integer program, where x_S is a 0/1 variable indicating if S is in $\mathcal{T}$.

P2. MIN SET COVER *Integer Program*

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{T}} c(S)\, x_S \\ \text{s.t.} \quad & \sum_{S : v \in S} x_S \;\geq\; 1 \qquad v \in V \\ & x_S \;\in\; \{0,1\} \qquad (3.3) \end{aligned}$$

We can transform **P2** to an linear program by relaxing (3.3) to instead read $x_S \in [0,1]$. Then we can solve the linear program to obtain the optimal relaxed solution $\mathbf{x}$. This solution consists of a real number x_S between 0 and 1 for each set

$S \in \mathcal{S}$. We of course need to know definitively which sets to include in our solution $\mathcal{T}$. To achieve this, we must round the $\mathbf{x}$ to a integral solution.

3.1.1 Rounding

Whenever we solve a continuous version of a combinatorial optimisation problem, we get a continuous solution. This solution is an optimal solution to the continuous problem, and thus has objective value as good as, if not better than, the combinatorial problem (as the combinatorial solutions are potential solutions to the continuous problem). However, it is unlikely that the continuous solution is a combinatorial solution.

To get a good combinatorial solution, we must somehow find a combinatorial solution which is *close* to the continuous solution. We must *round* the values of the continuous solution in order to a feasible combinatorial solution. For instance, in the Min Set Cover problem, our linear program will give us a vector $\mathbf{x}$, where $0 \leq x_S \leq 1$ for each set $S \in \mathcal{S}$. But for a combinatorial solution, we need a discrete selection of subsets, not a fractional one.

We cannot round $\mathbf{x}$ in any old fashion—for example, 'let S be in $\mathcal{T}$ if $x_S \geq 1/2$'—as then there is no guarantee that the $\mathcal{T}$ that we produce will be a feasible solution to the Min Set Cover problem. We must be a little smarter.

Suppose that f is the number of sets that the most frequently occurring element $v \in V$ appear in. Then we can use the following rounding technique, which is only slightly smarter than the naive version written above, yet gives us a f-approximation [104]:

Let $\mathcal{T} = \{$ all S such that $x_S \geq 1/f \}$.

The issue of how to round a continuous program is of fundamental importance when designing algorithms which involved relaxations.

3.2 Semi-Definite Programming

Although linear programming is the obvious relaxation to use when the combinatorial problem we are interested in is a integer program, it is common for the constraints not to be quite so simple. The constraints that **P1** allows are linear in each

variable—this only allows a certain amount of expressiveness about combinations of variables.

However, as we have previously noticed with the problems that we are studying in this work, the constraints that we are worried about for graph problems (the advice) usually involve two variables at once. So rather than a polytope (which is the shape in $\mathbb{R}^n$ formed by linear constraints), we have some other kind of feasible region (formally a cone). Although we could make further relaxations, and work in some polytope that contains the cone, we generally like to relax things as little as possible (for accuracy).

To deal with such problems, we can often use a relaxation to a Semi-Definite Program (SDP). Any SDP can be written in the following form, where $\mathbf{Y}$ is a $n \times n$ symmetric matrix:

P3. *Semi-Definite Programming*

$$\min \quad \mathbf{C} \bullet \mathbf{Y}$$

$$\begin{aligned} \text{s.t.} \quad \mathbf{A^i} \bullet \mathbf{Y} &= b_i, \quad i = 1, \ldots, m \qquad (3.4) \\ \mathbf{Y} &\succeq 0 \qquad (3.5) \end{aligned}$$

Here, the $\mathbf{A^i}$ and $\mathbf{C}$ are $n \times n$ symmetric matrices, and $\mathbf{b} \in \mathbb{R}^m$. The $\mathbf{A^i}$ and $\mathbf{b}$ form the constraints on the feasible region of the SDP, and the $\mathbf{C}$ matrix encodes the cost function. The inner product between such $n \times n$ symmetric matrices is the *trace inner product*:

$$\mathbf{X} \bullet \mathbf{Y} = \text{trace}(\mathbf{X}^T\mathbf{Y}) = \sum_{i,j} x_{ij} y_{ij}$$

This is essentially the standard Euclidean inner product, treating the matrix as one large vector in $\mathbb{R}^{n^2}$. In fact, if condition (3.5) instead read $\mathbf{Y} \geq 0$, then **P3** would be completely equivalent to **P1**. However, (3.5) says that $\mathbf{Y} \succeq 0$, that is $\mathbf{Y}$ is *Positive Semi-Definite*:

Definition 11. *A symmetric matrix* $\mathbf{Y}$ *is* Positive Semi-Definite *(PSD) if all eigenvalues of* $\mathbf{Y}$ *are non-negative. If all eigenvalues are positive,* $\mathbf{Y}$ *is* Positive Definite.

SDPs can also be solved in polynomial time, here in time $\tilde{O}(l^{3.5})$, where again l is a measure of the problem size (the number of variables and the number of

constraints).

Our main interest in positive semi-definite matrices comes from the Cholesky decomposition. This tells us that we can write any PSD matrix $\mathbf{Y}$ in terms of some set of n vectors ($\mathbf{x_i}$) in $\mathbb{R}^k$ (where $k \leq n$ is the rank of $\mathbf{Y}$):

$$y_{ij} = \mathbf{x_i}^T \mathbf{x_j} \tag{3.6}$$

Or equivalently, if $\mathbf{X} = [\mathbf{x_1} \cdots \mathbf{x_k}]^T$ (so $\mathbf{x_i}$ is the ith row of $\mathbf{X}$), then

$$\mathbf{Y} = \mathbf{X}\mathbf{X}^T$$

So, with this decomposition in mind, the objective function and constraints in **P3** can be thought of a functions of various inner products of the k-dimensional vectors $\mathbf{x_i}$. For instance, the value c_{ij} is now the coefficient of $\mathbf{x_i}^T \mathbf{x_j}$. So **P3** can take the form:

P4. *Semi-Definite Programming II*

$$\begin{aligned} \min \quad & \sum_{i,j \leq n} c_{ij} \mathbf{x_i}^T \mathbf{x_j} \\ \text{s.t.} \quad & \sum_{i,j \leq n} a_{ij}^{\ell} \mathbf{x_i}^T \mathbf{x_j} \;=\; b_\ell, \quad \ell = 1, \ldots, m \end{aligned} \tag{3.7}$$

3.2.1 Laplacian Matrices

The sums in **P4** are especially interesting when the matrices involved are *Laplacian* matrices. A Laplacian matrix is a generalisation of the *graph Laplacian*, a matrix which encodes many fundamental properties of a weighted graph. Suppose that $\mathbf{A}$ is a matrix representing the affinities between the vertices for a Min-Cut-style problem. So $a_{ij} = w(i, j)$. Then $\mathbf{A}$ is symmetric, with 0s on the diagonal, and we can define the graph Laplacian of G as a matrix $\mathcal{L}$, where

$$\mathcal{L}_{ij} = \begin{cases} \deg(i) & \text{if } i = j, \\ -a_{ij} & \text{otherwise.} \end{cases} \tag{3.8}$$

That is, $\mathcal{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D} = \text{Diag}(\deg(i))$, a $n \times n$ diagonal matrix consisting of the degrees of the vertices of G. We can generalise this concept by changing the underlying matrix from $\mathbf{A}$ to some other symmetric matrix:

Definition 12. *Let* $\mathbf{X}$ *be a symmetric* $n \times n$ *matrix. Then the Laplacian of* $\mathbf{X}$*,* $\mathcal{L}(\mathbf{X})$*, is defined:*

$$\mathcal{L}(\mathbf{X}) = \mathit{Diag}(\mathbf{X}\,\mathbf{1}) - \mathbf{X}\,,$$

where $\mathbf{1} = [1, \ldots, 1]^T$ *is the* n*-dimensional vector of ones.*

Laplacian matrices have some important properties, which are standard results in the literature [23]:

Remark 2. *For any vector* $\mathbf{x}$*, and symmetric matrix* $\mathbf{A}$*, of appropriate dimensions*

$$\mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} = \sum_{i<j} a_{ij}(x_i - x_j)^2$$

Proof.

$$\begin{aligned}
\mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} &= \mathbf{x}^T \left[\text{Diag}(\mathbf{A}\,\mathbf{1}) - \mathbf{A}\right]\mathbf{x} \\
&= \sum_i \Big(\sum_j a_{ij}\Big) x_i^2 - \sum_{i,j} a_{ij} x_i x_j \\
&= \sum_{i,j} a_{ij} x_i (x_i - x_j) \\
&= \frac{1}{2}\sum_{i,j} a_{ij} x_i (x_i - x_j) + \frac{1}{2}\sum_{j,i} a_{ij} x_j (x_j - x_i) \qquad \text{as } \mathbf{A} \text{ is symmetric} \\
&= \frac{1}{2}\sum_{i,j} a_{ij} (x_i - x_j)^2 \\
&= \sum_{i<j} a_{ij} (x_i - x_j)^2
\end{aligned}$$

□

Corollary 1. *For any symmetric* $\mathbf{A}$*,* $\mathcal{L}(\mathbf{A})$ *is positive semi-definite, with smallest eigenvalue* 0*, with corresponding eigenvector* $\mathbf{1}$*.*

Corollary 2. *Let* $\mathbf{Y}$ *be the Gram matrix of a set of* k *vectors* $\mathbf{x_i}$*, as in (3.6) above. Then*

$$\mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} = \sum_{i<j} a_{ij} \|\mathbf{x_i} - \mathbf{x_j}\|^2$$

Proof. Let $\mathbf{x^k}$ be the kth column of X. Then, by the properties of the trace,

$$\begin{aligned}
\mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} &= \text{trace}(\mathcal{L}(\mathbf{A})^T \mathbf{X}\mathbf{X}^T) \\
&= \text{trace}(\mathbf{X}^T \mathcal{L}(\mathbf{A})^T \mathbf{X}) \\
&= \sum_k \mathbf{x^k}^T \mathcal{L}(\mathbf{A}) \mathbf{x^k} \\
&= \sum_{k,i<j} a_{ij}(x_i^k - x_j^k)^2 \\
&= \sum_{i<j} a_{ij} \sum_k (x_i^k - x_j^k)^2 \\
&= \sum_{i<j} a_{ij} \|\mathbf{x_i} - \mathbf{x_j}\|^2
\end{aligned}$$

□

Example: Goemans Williamson SDP for Max-Cut

Let us now consider a concrete example of this Laplacian property in action; the classic Max-Cut SDP of Goemans and Williamson [54]. If $\mathbf{A}$ is a matrix of edge weights, and $x_i = \mathcal{C}(v_i)$ is the cluster membership for some clustering $\mathcal{C}$ (remembering that $\mathcal{C}(v_i) = \pm 1$), then the Max-Cut problem can be written:

P5. Max-Cut

$$\begin{aligned}
\max \quad & \frac{1}{4} \sum_{i<j} a_{ij}(x_i - x_j)^2 \\
\text{s.t.} \quad & \|x_i\| = 1 \\
& x_i \in \mathbb{R} \qquad (3.9)
\end{aligned}$$

If we now relax the constraint (3.9) and instead allow $\mathbf{x_i} \in \mathbb{R}^k$ for some $k \leq n$, then we have the following SDP:

P6. MAX-CUT *SDP*

$$\begin{aligned} \max \quad & \frac{1}{4}\mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} \\ \text{s.t.} \quad & y_{ii} = 1 \qquad (3.10) \\ & \mathbf{Y} \succeq 0 \end{aligned}$$

The constraint (3.10) can easily be written in the form of (3.4); we set $b_i = 1$, for $i = 1$ to n, and $\mathbf{A^i} = \mathbf{E_{ii}}$, the matrix with 1 in position (i, i) and zeroes elsewhere. SDPs, similarly to LPs, can be solved in polynomial time (see e.g. [55]). So an optimal continuous solution $\mathbf{Y}$ to **P6** can be found; this can be decomposed to a unit vector $\mathbf{x}_i$, for each point $v_i \in V$.

Goemans and Williamson rounded those vectors ($\mathbf{x_i}$) to a clustering for the vertices V, in a fashion which is now a standard technique, known as *random hyperplane* rounding. We simply choose some vector $\mathbf{r}$ in $\mathbb{R}^n$, and set $\mathcal{C}(i) = 1$ iff $\mathbf{r}^T\mathbf{x_i} \geq 0$. This method yields a 0.878-approximation to the general MAX-CUT problem. Moreover, assuming the unique games conjecture, this is essentially the best possible [72] for this problem.

3.3 Spectral Clustering

When relaxing a combinatorial problem, such as an integer program, to a continuous problem, such as a linear program, we take advantage of the polynomial time solvability of the continuous problem, sacrificing solution accuracy in the underlying combinatorial problem. However, LPs and SDPs are very general problems—there exist problems which can be relaxed to very specific type of continuous problems.

SPECTRAL CLUSTERING is a relaxation of AFFINITY CLUSTERING problems to *eigenvalue* problems. Such problems can be solved exactly by numerical analysis, an area which has many highly-efficient and scalable algorithms. For that reason—along with the high quality of real-world results—SPECTRAL CLUSTERING is an important clustering technique.

Let us consider the RATIO CUT problem. As we did with the MAX-CUT problem above, if we are clustering into two clusters, we can represent any clustering $\mathcal{C}$ by a

vector $\mathbf{x} \in \{\pm 1\}^n$—let $x_i = \mathcal{C}(v_i)$. Remember that in the two cluster case,

$$\textsc{Ratio Cut}(\mathcal{C}) = \frac{\text{cut}(C_1, C_2)}{|C_1|} + \frac{\text{cut}(C_2, C_1)}{|C_2|} = n\frac{\text{cut}(C_1, C_2)}{|C_1||C_2|}$$

Then, the objective function can be written in terms of $\mathbf{x}$, using Remark 2:

$$\textsc{Ratio Cut}(\mathbf{x}) = n\frac{\sum_{i<j} a_{ij}(x_i - x_j)^2}{\sum_{i<j}(x_i - x_j)^2} = n\frac{\mathbf{x}^T\mathcal{L}(\mathbf{A})\mathbf{x}}{\mathbf{x}^T\mathcal{L}(\mathbf{1})\mathbf{x}} \tag{3.11}$$

This is due of course to the fact that

$$(x_i - x_j)^2 = \begin{cases} 4 & \text{if } \mathcal{C}(v_i) \neq \mathcal{C}(v_j), \\ 0 & \text{otherwise.} \end{cases}$$

We now have two options for a relaxation. The first is similar to the GW relaxation of Section 3.2.1; to relax x_i from a one-dimensional vector in $\mathbb{R}$ to a full vector in some $\mathbb{R}^k$, whilst maintaining the property that $\|\mathbf{x_i}\| = 1$. This leads to a similar SDP to **P6**, namely the following, similar to that of De Bie and Cristianini [31]:

P7. Ratio Cut *SDP*

$$\begin{aligned} \min \quad & \mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} \\ \text{s.t.} \quad & \mathcal{L}(\mathbf{1}) \bullet \mathbf{Y} = 1 \qquad (3.12) \\ & y_{ii} = q \qquad (3.13) \\ & \mathbf{Y} \succeq 0 \end{aligned}$$

The constraint (3.12) has been added and the relevant term removed from the denominator as the original problem is *scaling invariant*—that is, any two scaled solutions $\mathbf{x}$ and $q\mathbf{x}$ will have the same value in (3.11)—so there must be a optimal solution which satisfies (3.12).

3.3.1 The Spectral Relaxation

In the SDP relaxation, we relaxed each co-ordinate of $\mathbf{x}$ to be in $\mathbb{R}^k$ rather than $\mathbb{R}$, whilst maintaining the property that $\|\mathbf{x_i}\| = 1$. (Thus we gained a matrix

$\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_n]^T$, which led us to the PSD matrix $\mathbf{Y} = \mathbf{X}\mathbf{X}^T$).

In the Spectral relaxation, we instead leave x_i in the one-dimensional space $\mathbb{R}$, but remove the constraint $\|x_i\| = 1$. So instead of taking the values ± 1, x_i can take some value in $\mathbb{R}$. This is equivalent to removing the constraint (3.13) from **P7**, and leads to the following problem:

P8. *Un-normalised Spectral Clustering (*Ratio Cut*)*

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} \\ \text{s.t.} \quad & \\ & \mathbf{x}^T \mathcal{L}(\mathbf{1})\mathbf{x} = 1 \end{aligned} \tag{3.14}$$

This problem can be modified slightly by adding the redundant constraint $\mathbf{x}^T\mathbf{1} = 0$—we can assume this as $\mathbf{x}^T\mathcal{L}(\mathbf{A})\mathbf{x}$ is invariant under translation (refer to Remark 2). Then **P8** becomes:

P9. *Un-normalised Spectral Clustering (*Ratio Cut*) II*

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{x}^T\mathbf{1} = 0 \\ & \|\mathbf{x}\|^2 = 1/n \end{aligned} \tag{3.15}$$

The constraint (3.15) follows as

$$\mathbf{x}^T \mathcal{L}(\mathbf{1})\mathbf{x} = \mathbf{x}^T \left(n\mathbf{I} - \mathbf{1}\right) \mathbf{x} = n\|\mathbf{x}\|^2$$

As we know (Corollary 1) that $\mathbf{1}$ is the eigenvector of $\mathcal{L}(\mathbf{A})$ with smallest eigenvalue, the Rayleigh-Ritz theorem [93] tells us the solution to **P9** is given by the second smallest eigenvector of $\mathcal{L}(\mathbf{A})$—scaled to satisfy (3.15). This vector is well studied—it is known as the Fiedler [45] vector. So we can find the exact solution to minimise the Ratio Cut objective in the relaxation.

Of course the Fiedler vector is a relaxed solution; we need to round it to produce a feasible solution to the Ratio Cut problem. Fortunately, in the two cluster case, rounding is straightforward. As the value x_i indicates cluster membership (remember in the combinatorial problem $x_i = \pm 1$, depending on cluster membership), it

makes sense to order the vertices by x_i value, and split the list into two clusters at some mid-point.

There are $n-1$ choices of such mid-points, and it is a simple matter to test the RATIO CUT cost of each choice. We can then choose the minimum partition that we have found. Another simpler rounding technique is simply to cut at zero—place all points with $x_i \geq 0$ into one cluster, and other points into another. This is the technique that we will use in our experiments.

3.3.2 NORMALISED CUT

The NORMALISED CUT problem is similar to the RATIO CUT problem. We will sketch a similar construction of NORMALISED CUT SDP and spectral relaxations, both of which follow in a fashion similar to the RATIO CUT relaxations given above.

Remembering that, for two clusters, $\text{vol}(X) = \sum_{v \in X} \deg(v)$, and

$$\text{NORMALISED CUT}(\mathcal{C}) = \frac{\text{cut}(C_1, C_2)}{\text{vol}(C_1)} + \frac{\text{cut}(C_2, C_1)}{\text{vol}(C_2)} = \text{vol}(V)\frac{\text{cut}(C_1, C_2)}{\text{vol}(C_1)\,\text{vol}(C_2)}\,,$$

we produce an analogy of (3.11), where $\mathbf{d} = [\deg(v_1), \ldots, \deg(v_n)]^T$:

$$\begin{aligned}\text{NORMALISED CUT}(\mathbf{x}) &= \text{vol}(V)\frac{\sum_{i<j} a_{ij}(x_i - x_j)^2}{\sum_{i<j} \deg(v_i)\deg(v_j)(x_i - x_j)^2} \\ &= \text{vol}(V)\frac{\mathbf{x}^T\mathcal{L}(\mathbf{A})\mathbf{x}}{\mathbf{x}^T\mathcal{L}(\mathbf{dd^T})\mathbf{x}}\,.\end{aligned}$$

Again, if we relax $\mathbf{x_i}$ to be in $\mathbb{R}^k$, we get, completely analogously to **P7**,

P10. NORMALISED CUT *SDP*

$$\begin{aligned}\min \quad & \mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} \\ \text{s.t.} \quad & \mathcal{L}(\mathbf{dd^T}) \bullet \mathbf{Y} = 1 \\ & y_{ii} = q \\ & \mathbf{Y} \succeq 0\end{aligned}$$

If we instead use the spectral relaxation of **P8**, we obtain the following,

P11. *Normalised Spectral Clustering (*NORMALISED CUT*)*

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{x}^T \mathcal{L}(\mathbf{d}\mathbf{d}^\mathbf{T})\mathbf{x} = 1 \end{aligned}$$

This time, we add the redundant constraint $\mathbf{x}^T\mathbf{d} = 0$, again as the objective is invariant under translation, and as

$$\mathcal{L}(\mathbf{d}\mathbf{d}^\mathbf{T}) = \mathrm{vol}(V)\,\mathbf{D} - \mathbf{d}\mathbf{d}^T,$$

we get,

P12. *Normalised Spectral Clustering (*NORMALISED CUT*) II*

$$\begin{aligned} \min \quad & \mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} \\ \text{s.t.} \quad & \mathbf{x}^T\mathbf{d} = 0 \\ & \mathbf{x}^T\mathbf{D}\mathbf{x} = \frac{1}{\mathrm{vol}(V)} \end{aligned}$$

This time the Rayleigh-Ritz theorem tells us the solution to **P12** is given by $\mathbf{x} = \mathbf{D}^{\mathbf{1/2}}\mathbf{u}$, where $\mathbf{u}$ is the second smallest eigenvector of $\mathbf{D}^{-\mathbf{1/2}}\mathcal{L}(\mathbf{A})\mathbf{D}^{-\mathbf{1/2}}$, also PSD. This is once again simple to calculate using numerical analysis packages.

3.4 Relaxing Affinity Clustering with Advice

In this chapter we are interested in using the relaxations that we have mentioned to solve the AFFINITY CLUSTERING WITH ADVICE problem, with the restriction of two clusters. So we will be transforming a combinatorial clustering problem into a relaxed continuous problem. The main difficulty here is a lack of a clear objective function.

We are essentially trying to solve two problems simultaneously: a AFFINITY CLUSTERING problem (defined on an *affinity graph*) and MIN-2-CC (defined on an *advice graph*), as we know that the advice that we have been given is not necessarily correct, or necessarily even consistent. A common approach is to modify the objec-

tive function to respect the advice, however—particularly for the modified version of Spectral Clustering considered by Kamvar et al. [63]—it is not at all clear why the particular modification is justified.

If we knew that the advice was correct (we treated the advice as *constraints*), then we could use the following 'subspace trick' of De Bie, Suykens and De Moor [32] to restrict our relaxed search to be in a subspace containing only solutions that agree with the constraints.

3.4.1 The Subspace Trick

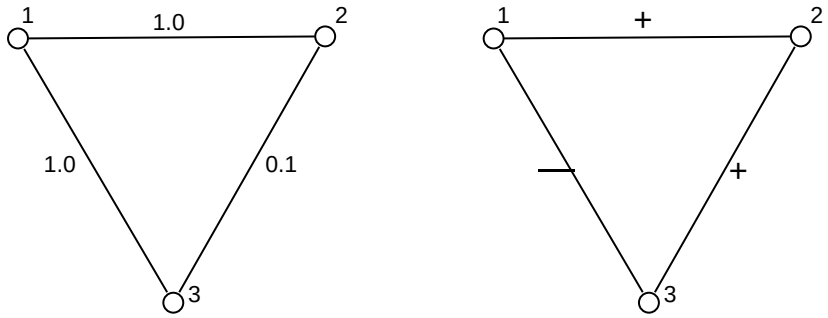
The subspace trick of De Bie, Suykens and De Moor [32] gives a method for incorporating consistent advice into spectral and SDP relaxations of Normalised Cut. As an example, consider spectral clustering and suppose we have two 'blocks' of independent advice. Suppose we think that two vertices, say v_1 and v_2, should be in the same cluster and both should be in a different cluster to v_3. Additionally, maybe we think that vertices v_4 and v_5 should be in the same cluster.

Considering this advice, we could pass this advice in the form of a constraint to our relaxed problem, by requiring the solution vector $\mathbf{x}$ to have $x_1 = x_2$ and $x_4 = x_5$—guaranteeing that these pairs of vertices end up in the same cluster after rounding. It also makes sense to constrain $\mathbf{x}$ so that $x_3 = -x_2 = -x_1$. This can be done by assuming $\mathbf{x}$ (our solution vector) has the form

$$\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \mathbf{I_{n-5}} \end{bmatrix} \mathbf{z}$$

where $\mathbf{z} \in \mathbb{R}^{n-3}$. We have a large identity matrix corresponding to the vertices of which we have no advice. So we can rephrase the spectral clustering problem in terms of $\mathbf{z}$ and ensure that the advice holds. We defer the details of this process to Section 3.6.3.

This technique leaves the spectral algorithm essentially unchanged; it now just



(a) NORMALISED CUT problem

(b) CORRELATION CLUSTERING problem

Figure 3.1: A problem for which not all optimal solutions to MIN-2-CC are optimal for the accompanying NORMALISED CUT problem.

searches for eigenvectors in a different subspace. However it is not necessarily apparent from their work how to extend this technique to inconsistent advice. *This is the key issue addressed in this chapter.*

3.4.2 Addressing Inconsistency

So how can we apply the subspace trick when the advice we have is no longer consistent?

As a first approach (NAIVE), we could simply try to solve the MIN-2-CC problem defined by the advice, and reject advice that this solution fails to respect. A good solution to MIN-2-CC will ensure that we minimise the number of such edges that we will have to ignore. Then the advice that remains will be consistent, and we can then use the subspace trick. Or indeed, we could use any other constraint-based clustering algorithm in this way.

However, this idea has some problems. A toy example of inconsistent advice is shown in Figure 3.1. We can see that deleting any one of the three edges will result in an optimal solution to the MIN-2-CC problem, leading the the three solutions $\{\{1,2,3\},\emptyset\}$, $\{\{1\}, \{2,3\}\}$ and $\{\{1,2\}, \{3\}\}$. However, one specific cut (namely separating vertex 3 from vertices 1 and 2) has a much better NORMALISED CUT cost. So, in forcing a particular optimal solution to MIN-2-CC, we are constraining our NORMALISED CUT solution too much.

A second approach (Cost-Constrained) that solves this problem is to calculate the *cost* of an approximately optimal solution to Min-2-CC. Rather than force our Normalised Cut solution to be consistent with this Min-2-CC solution, instead we simply require that our Normalised Cut solution has the same correlation clustering cost. This approach will give the Normalised Cut side of our algorithm some room to move, avoiding situations like Figure 3.1. This technique will be outlined in Section 3.6.1.

A third approach (Cost-Bounded) is to allow the algorithm to differ from the optimum Correlation Clustering cost, but only by some a given factor. So now the Normalised Cut side of the problem has some breathing space in which to find a good solution, whilst we are still forcing a very good solution to Min-2-CC. This approach is developed in Section 3.6.2.

3.5 Relaxations of Correlation Clustering.

In this section we will consider the Correlation Clustering problem (specifically Min-2-CC) in the context of relaxations. This will allow us to relax Correlation Clustering to continuous versions analogous to the SDP and Spectral Clustering solutions to the Affinity Clustering problems (**P7** and **P10**, and **P9** and **P12**), which will allow us to combine the two problems in a sensible way.

3.5.1 Min-2-CC — the Combinatorial Problem

In general, unlike the affinity graph, the advice graph is not connected. So we can solve Correlation Clustering independently on each connected component. We call the vertices in a connected component an *advice block*. We assume without loss of generality that the order on the vertices ensures that the vertices within each block are consecutive. Here we will deal with the problem of solving Min-2-CC for a single advice block $\mathcal{B}$ with m vertices. In later sections, we will consider multiple advice blocks.

Again, we assign $x_i = \pm 1$ to each vertex depending on the cluster in which we place that vertex. For convenience, let $\mathbf{E_{ij}}$ be the matrix with a 1 in the (i, j) entry and zeros everywhere else.

Our immediate aim is to find, in terms of $\mathbf{x}$, a simple expression for the number of constraints violated by the labelling. Consider a single edge $e = \{i, j\}$ of $\mathcal{B}$ with label $l(e)$. Define

$$\mathbf{M_e} = (\mathbf{E_{ii}} + \mathbf{E_{jj}}) - l(e)(\mathbf{E_{ij}} + \mathbf{E_{ji}})$$

and note that $\mathbf{M_e} \succeq 0$ because its eigenvalues are 0 and 2. Now

$$\begin{aligned}\mathbf{x}^T\mathbf{M_e}\mathbf{x} &= x_i^2 - 2l(e)x_ix_j + x_j^2 \\ &= |x_i - l(e)x_j|^2 \\ &= \begin{cases} 0 & \text{if } \mathbf{x} \text{ respects the advice on } e \\ 4 & \text{otherwise.} \end{cases}\end{aligned}$$

So if we define $\mathbf{M}_\mathcal{B} = \sum_e \mathbf{M_e}$ it follows that

$$\mathbf{x}^T\mathbf{M}_\mathcal{B}\mathbf{x} = 4 \times (\#\text{ pieces of advice violated by } \mathbf{x}).$$

Thus MIN-2-CC is essentially

$$\min_{\mathbf{x}\in\{-1,1\}^m} \mathbf{x}^T\mathbf{M}_\mathcal{B}\mathbf{x}\,. \tag{3.16}$$

Note that a clustering that satisfies all the advice in a block will have cost zero. Also observe that $\mathbf{x}^T\mathbf{x} = m$ is a constant so we could replace the objective function with $(\mathbf{x}^T\mathbf{M}_\mathcal{B}\mathbf{x})/(\mathbf{x}^T\mathbf{x})$ without changing the optimum vector.

3.5.2 Relaxations of MIN-2-CC

Recall that our overall aim is to constrain any algorithm we have for (approximately) solving NORMALISED CUT to produce clusterings which are, in terms of MIN-2-CC cost, not much worse than the optimum.

Since we cannot hope to solve (3.16) exactly, we will instead solve a relaxed version of it. We consider two relaxations which arise in much the same way as the spectral and SDP relaxations of NORMALISED CUT.

P13. *Spectral relaxation of correlation clustering*

$$\min_{\mathbf{x}\in\mathbb{R}^n} \quad \frac{\mathbf{x}^T\mathbf{M}_{\mathcal{B}}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$$

Observe that the solution of **P13** is given by any non-zero vector in the λ_{min}-eigenspace of $\mathbf{M}_{\mathcal{B}}$.

P14. *SDP relaxation of correlation clustering*

$$\begin{aligned} \min_{\mathbf{Y}} \quad & \mathbf{M}_{\mathcal{B}} \bullet \mathbf{Y} \\ \text{s.t.} \quad \forall\, i \in [m] \quad & y_{ii} = 1 \qquad (3.17) \\ & \mathbf{Y} \succeq 0 \end{aligned}$$

For either relaxation, if the advice is consistent, the relaxation produces a solution of the same cost (zero) as the optimal solution to the combinatorial problem (3.16). This is because any solution of the original problem is a feasible point of the relaxed problem, and the relaxed problem has non-negative cost as $\mathbf{M}_{\mathcal{B}} \succeq 0$.

3.6 Affinity Clustering with Advice

In this section, we give the details of Cost-Constrained and Cost-Bounded, introduced in Section 3.4.2, for both the spectral and SDP relaxations of Normalised Cut.

Throughout, let $\mathcal{B}_1, \ldots, \mathcal{B}_p$ be the advice blocks of the advice graph. Let $\mathbf{x}_{\mathcal{B}}$ denote the projection of $\mathbf{x}$ onto the co-ordinates involved in advice block $\mathcal{B}$. Along similar lines, if $\mathbf{z}_{\mathcal{B}}$ is a vector of length $|\mathcal{B}| \leq n$ associated with the advice block $\mathcal{B}$, define $\widetilde{\mathbf{z}_{\mathcal{B}}}$ to be the length-n vector that agrees with $\mathbf{z}_{\mathcal{B}}$ in the appropriate co-ordinates and has zeros elsewhere. For a $|\mathcal{B}| \times |\mathcal{B}|$ matrix $\mathbf{M}_{\mathcal{B}}$ we also define $\widetilde{\mathbf{M}_{\mathcal{B}}}$ in a similar fashion.

3.6.1 Cost-Constrained

Let OPT_j denote the optimum cost of the SDP relaxation of Min-2-CC (**P14**) for block j. For the SDP relaxation, we can add the constraint that for each advice

block, the MIN-2-CC cost of point $\mathbf{Y}$ is at most $q \cdot \text{OPT}_j$. (The scaling by q is necessary because in **P14** the variables satisfy $y_{ii} = 1$ whereas in **P10** the variables satisfy $y_{ii} = q$.) This forces the new SDP (**P15**) only to consider points of minimum SDP-relaxed MIN-2-CC cost.

P15. COST-CONSTRAINED *(SDP version)*

$$\begin{aligned}
\min_{\mathbf{Y},q} \quad & \mathcal{L}(\mathbf{A}) \bullet \mathbf{Y} \\
& \mathcal{L}(\mathbf{d}\mathbf{d}^{\mathbf{T}}) \bullet \mathbf{Y} = 1 \\
\text{s.t.} \quad & \forall i \in [n] \quad y_{ii} = q \\
& \forall j \in [p] \quad \widetilde{\mathbf{M}_{\mathcal{B}_\mathbf{j}}} \bullet \mathbf{Y} \leq q \cdot \text{OPT}_j \qquad (3.18) \\
& \mathbf{Y} \succeq 0
\end{aligned}$$

In the spectral case, the analogous step would be to add the following constraints to the spectral relaxation of NORMALISED CUT.

$$\forall j \in [p] \quad \frac{\mathbf{x}_{\mathcal{B}_\mathbf{j}}{}^T \mathbf{M}_{\mathbf{B_j}} \mathbf{x}_{\mathcal{B}_j}}{\mathbf{x}_{\mathcal{B}_\mathbf{j}}{}^T \mathbf{x}_{\mathcal{B}_j}} \leq \lambda_{min}(\mathbf{M}_{\mathcal{B}_\mathbf{j}}) \qquad (3.19)$$

But doing so would mean the problem would no longer be an eigenvalue problem—in fact it would be an SDP—which would undermine the main strength of spectral clustering, its speed.

Luckily, the condition in (3.19) is equivalent to the condition that each $\mathbf{x}_{\mathcal{B}_j}$ is in the λ_{min}-eigenspace of $\mathbf{M}_{\mathcal{B}_\mathbf{j}}$, resulting in **P16**.

P16. COST-CONSTRAINED *(spectral version)*

$$\begin{aligned}
\min_{v} \quad & \frac{\mathbf{x}^T \mathcal{L}(\mathbf{A}) \mathbf{x}}{\mathbf{x}^T \mathbf{D} \mathbf{x}} \\
\text{s.t.} \quad & \mathbf{d}^T \mathbf{x} = 0 \qquad (3.20) \\
& \forall j \in [p] \quad \mathbf{x}_{\mathcal{B}_j} \in \lambda_{min}\text{-eigenspace of } \mathbf{M}_{\mathcal{B}_\mathbf{j}} \qquad (3.21)
\end{aligned}$$

The constraints (3.20) and (3.21) are forcing $\mathbf{x}$ to be in some linear subspace of $\mathbb{R}^n$. So the problem can then be solved using the subspace trick, as we will see in Section 3.6.3.

3.6.2 Cost-Bounded

The main drawback of Cost-Constrained is that it does not give the algorithm much freedom to balance the trade-off between the Min-2-CC and the Normalised Cut problem. If the advice is quite inconsistent, then forcing the algorithm to follow solutions of a relaxation of Min-2-CC too closely might result in poor performance.

Above, we forced the algorithm to produce a (relaxation of) a clustering whose cost was at most the minimum cost of the appropriate relaxation of Min-2-CC. Now we introduce a parameter $f \geq 1$ which tells us the factor by which we are willing to exceed the (relaxed optimal) Min-2-CC cost.

This is straightforward to introduce to the SDP formulation. We simply replace the constraints (3.19) of **P15** with

$$\forall j \in [p] \quad \widetilde{\mathbf{M}_{\mathcal{B}_j}} \bullet \mathbf{Y} \leq f \cdot q \cdot \text{OPT}_j. \tag{3.22}$$

In the spectral formulation, the constraints we actually want to add are

$$\forall j \in [p] \quad \frac{\mathbf{x}_{\mathcal{B}_j}{}^T \mathbf{M}_{\mathbf{B}_j} \mathbf{x}_{\mathcal{B}_j}}{\mathbf{x}_{\mathcal{B}_j}{}^T \mathbf{x}_{\mathcal{B}_j}} \leq f \cdot \lambda_{min}(\mathbf{M}_{\mathcal{B}_j}) \tag{3.23}$$

but, again we cannot add these and still have an eigenvalue problem. Unfortunately in this case we cannot get a constraint equivalent to (3.23) by the subspace trick. So, in the interests of producing a practical algorithm, we approximate (3.23) by

$$\forall j \in [p] \quad \mathbf{x}_{\mathcal{B}_j} \in (\leq f \cdot \lambda_{min})\text{-eigenspace of } \mathbf{M}_{\mathcal{B}_j} \tag{3.24}$$

where the $(\leq f \cdot \lambda_{min})$-eigenspace of $\mathbf{M}_{\mathcal{B}_j}$ is the span of all eigenvectors of $\mathbf{M}_{\mathcal{B}_j}$ with eigenvalue at most $f \cdot \lambda_{min}$. If $\mathbf{x}$ satisfies (3.24) then it satisfies (3.23), but the converse does not necessarily hold.

Replacing the constraints in (3.21) of **P16** with the constraints in (3.24) gives our final spectral algorithm for clustering with inconsistent advice. It can again be solved with the subspace trick, using the techniques outlined in the next section, because all the constraints simply force $\mathbf{x}$ to be in some linear subspace of $\mathbb{R}^n$.

3.6.3 Combining subspace constraints

In this section we will demonstrate how the subspace trick can be extended to deal with any subspace, specifically constraints of the form given by (3.20) and (3.21) or (3.24). These constraints are asking us to find a vector orthogonal to $\mathbf{d}$, whilst being within a certain subspace generated by particular eigenvectors of $\mathbf{M}_{\mathcal{B}_j}$.

Definition 13. *The* range *of a $n \times m$ matrix $\mathbf{A}$, $\mathcal{R}ange(\mathbf{X})$, is the subspace of $\mathbb{R}^n$ consisting of all vectors $\mathbf{x}$ such that $\mathbf{x} = \mathbf{A}\mathbf{y}$ for $\mathbf{y} \in \mathbb{R}^m$.*

The nullspace *of a $n \times m$ matrix $\mathbf{A}$, $\mathcal{N}ull(\mathbf{X})$, is the subspace of $\mathbb{R}^m$ consisting of all vectors $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} = 0$.*

Let $\mathbf{W}_{\mathcal{B}_j}$ be a matrix whose columns are a basis for the ($\leq f \cdot \lambda_{min}$)-eigenspace of $\mathbf{M}_{\mathcal{B}_j}$. Then the COST-BOUNDED version of **P16** can be written as follows (COST-CONSTRAINED simply sets $f = 1$):

P17. *Spectral clustering with inconsistent advice*

$$\begin{aligned}
\min_{\mathbf{x}} \quad & \mathbf{x}^T \mathcal{L}(\mathbf{A})\mathbf{x} \\
\text{s.t.} \quad & \mathbf{x}^T\mathbf{D}\mathbf{x} = 1 \\
& \mathbf{x} \in \mathcal{N}\text{ull}(\mathbf{d}^T) \qquad (3.25) \\
& \forall j \in [p] \quad \mathbf{x}_{\mathcal{B}_j} \in \mathcal{R}\text{ange}(\mathbf{W}_{\mathcal{B}_j}) \qquad (3.26)
\end{aligned}$$

Let

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_{\mathcal{B}_1} & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \mathbf{W}_{\mathcal{B}_p} & 0 \\ 0 & \cdots & 0 & \mathbf{I} \end{bmatrix},$$

where the dimension of $\mathbf{I}$ is the number of vertices not involved in any advice. Then we can replace the constraints (3.25) and (3.26) with

$$\mathbf{x} \in \mathcal{N}\text{ull}(\mathbf{d}^T) \cap \mathcal{R}\text{ange}(\mathbf{W}) = S \subseteq \mathbb{R}^n$$

Suppose $\mathbf{Z}$ is a matrix satisfying $\mathcal{R}\text{ange}(\mathbf{Z}) = S$ and $\mathbf{Z}^T\mathbf{D}\mathbf{Z} = I$. Then if we let

$\mathbf{x} = \mathbf{Z}\,\mathbf{z}$, for some $\mathbf{z}$, it is clear that $\mathbf{x} \in S$, and **P17** becomes

$$\min\ (\mathbf{z}^T\,\mathbf{Z}^T)\mathcal{L}(\mathbf{A})(\mathbf{Z}\,\mathbf{z}) \quad \text{s.t.} \quad \mathbf{z}^T\mathbf{z} = 1 \tag{3.27}$$

A solution of (3.27) is given by taking $\mathbf{z}$ to be an unit-eigenvector corresponding to the smallest eigenvalue of $\mathbf{Z}^T\mathcal{L}(\mathbf{A})\mathbf{Z}$. The solution to the original problem is then $\mathbf{x} = \mathbf{Z}\,\mathbf{z}$.

Generating Z We can calculate $\mathbf{Z}$ as follows:

1. Let $\mathbf{R}$ be a matrix whose columns are an orthonormal basis for $\mathcal{R}\text{ange}(\mathbf{W})$ with respect to the inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{D}} = \mathbf{y}^T\mathbf{D}\,\mathbf{x}$
2. Let $\mathbf{N}$ be a matrix whose columns are an orthonormal basis for $\mathcal{N}\text{ull}(\mathbf{d}^T\mathbf{R})$ with respect to the inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{y}^T\mathbf{x}$.
3. Set $\mathbf{Z} = \mathbf{RN}$.

Remark 3. *The matrix* $\mathbf{Z}$*, as specified above, satisfies* $\mathcal{R}ange(\mathbf{Z}) = \mathcal{N}ull(\mathbf{d}^T) \cap \mathcal{R}ange(\mathbf{W})$ *and* $\mathbf{Z}^T\mathbf{D}\mathbf{Z} = I$*.*

Proof. Let $\mathbf{x} = \mathbf{Z}\,\mathbf{z}$ be in the range of $\mathbf{Z}$. Then $\mathbf{x} = \mathbf{RN}\,\mathbf{z}$, so $\mathbf{x}$ is a linear combination of the columns of $\mathbf{R}$. As the columns of $\mathbf{R}$ are an orthonormal basis for $\mathcal{R}\text{ange}(\mathbf{W})$, it follows that $\mathbf{x} \in \mathcal{R}\text{ange}(\mathbf{W})$. Consider $\mathbf{d}^T\mathbf{x}$. By the definition of $\mathbf{N}$,

$$\mathbf{d}^T\mathbf{x} = \mathbf{d}^T\mathbf{RN}\,\mathbf{z} = 0\,.$$

Conversely, let $\mathbf{x} \in \mathcal{N}\text{ull}(\mathbf{d}^T) \cap \mathcal{R}\text{ange}(\mathbf{W})$. Then, as $\mathcal{R}\text{ange}(\mathbf{R}) = \mathcal{R}\text{ange}(\mathbf{W})$, there is a $\mathbf{r}$ such that $\mathbf{x} = \mathbf{R}\,\mathbf{r}$. Also $\mathbf{d}^T\mathbf{x} = 0$. Combining the two, we get:

$$\mathbf{d}^T\mathbf{R}\,\mathbf{r} = 0\,.$$

So $\mathbf{r} \in \mathcal{N}\text{ull}(\mathbf{d}^T\mathbf{R})$. So there is a $\mathbf{z}$ such that $\mathbf{r} = \mathbf{N}\,\mathbf{z}$. But then $\mathbf{x} = \mathbf{R}\,\mathbf{r} = \mathbf{RN}\,\mathbf{z}$, and thus $\mathbf{x} \in \mathcal{R}\text{ange}(\mathbf{Z})$.

Finally, the orthogonality of the columns of $\mathbf{R}$ mean that $\mathbf{R}^T\mathbf{DR} = \mathbf{I}$. Similarly, $\mathbf{N}^T\mathbf{N} = \mathbf{I}$, giving

$$\mathbf{Z}^T\mathbf{DZ} = \mathbf{N}^T\mathbf{R}^T\mathbf{DRN} = \mathbf{N}^T\mathbf{IN} = \mathbf{I}$$ □

3.7 Experimental Investigations

3.7.1 Experiment Setup

In order to test the performance of the algorithms on realistic datasets, we used six of the UCI repository datasets [10]. All datasets are multi-dimensional binary classification problems. So they represent a task of classifying a set of high-dimensional data to agree with an existing classification.

Both datasets were stripped of incomplete records, and in one case (the **Spambase** dataset), sampled down to 500 datapoints. In each case, the two underlying clusters were of different sizes, which contributed to the mediocre performance of the pure spectral algorithm. This gives us reason to believe that adding advice will help the situation.

For reasons of speed, our experiments primarily use the spectral version of each of the algorithms. Relaxed solutions are rounded to clusterings by cutting at zero. This ensures that advice respected in the relaxed solution is respected in the final clustering.

All experiments were run on a 2 GHz Intel Core 2 Duo machine with 2GB of RAM, running MAC OS X. All algorithms were run in Matlab 7.4.0.

Advice We generated two different 'types' of synthetic advice for these problems to learn how the algorithms perform. The first we call Dense, which involves around n pieces of advice. We concentrated advice within 5 separate groups of 20 datapoints. This simulates a few sets of experiments done on some small subset of the total dataspace. Each piece of advice agrees with the actual classification, independently, with some probability p.

The second type of advice is the Complete case—here we are simulating pairwise comparisons that are relatively cheap, but quite noisy. So we generate a piece of advice for each pair of datapoints, and thus our advice graph is complete.

Min-2-CC Algorithms In order to test Naive, we need to solve Min-2-CC on each advice block. In the Dense case, we use a tight, strongly performing SDP relaxation [1]. In the complete case, we use Pick-a-Vertex, the simple 3-approximation algorithm of Bansal, Blum and Chalwa [12] followed by an application

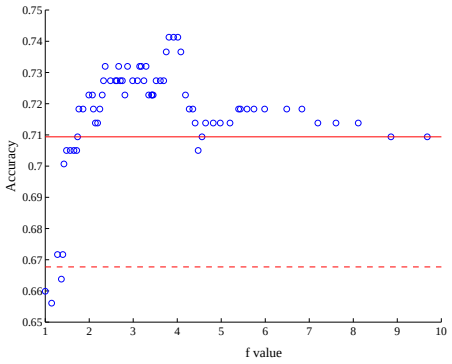


Figure 3.2: **Heart Disease** dataset, Dense advice, $p = 0.75$. The unbroken line is the baseline no advice accuracy; the dashed line is the correlation clustering based algorithm (Naive). Note that a naive algorithm can easily achieve a baseline accuracy of 0.54 for this dataset.

of Tosses (our local-search algorithm of Chapter 4, where we develop the PASTA-toss algorithm, along with other better algorithms for this part of the problem).

For each advice type on each dataset, we ran spectral clustering with no advice (as a baseline), Naive (as a second baseline), and then spectral clustering with every different meaningful f value from 1 upwards. This means every increment in f added one additional eigenvector to a single block, until all eigenvectors were added (which is exactly the same as the no advice case).

Accuracy The metric that we use to measure the effectiveness of each variant of our algorithm is *accuracy*—that is the proportion of points that are placed into the correct cluster. This means that the lowest possible value obtainable is 0.5. We note that a naive algorithm which simply places all points in one cluster will achieve an accuracy equal to the probability of any point being in the larger cluster. We indicate this baseline accuracy for each dataset.

3.7.2 Results

Dense advice Figure 3.2 displays the results of the Dense advice problem on the **Heart Disease** dataset with $p = 0.75$. We can see that the advice here is sufficiently inconsistent that algorithms which follow it closely (i.e. Naive and Cost-

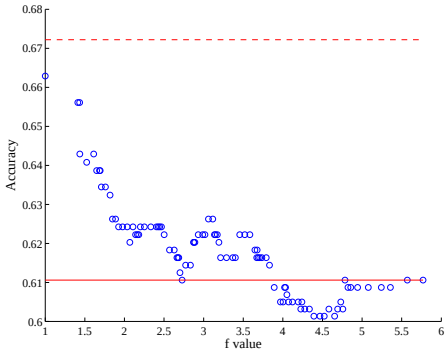


Figure 3.3: **Spambase** dataset, Dense Advice, $p = 0.75$. Note that a naive algorithm can easily achieve a baseline accuracy of 0.59 for this dataset.

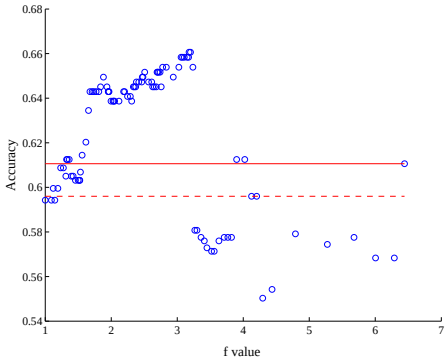


Figure 3.4: **Spambase** dataset, Dense advice, $p = 0.65$.

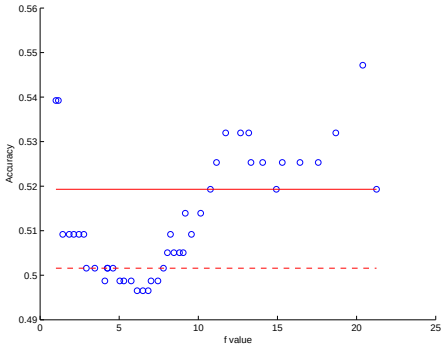


Figure 3.5: **Hepatitis** dataset, Dense advice, $p = 0.6$. Note that a naive algorithm can easily achieve a baseline accuracy of 0.84 for this dataset.

Constrained) perform far worse than the algorithm that ignores it completely (that is, spectral alone). But we can see that by increasing f and striking a balance between ignoring advice and respecting it too strongly, we can achieve results that outperform either extreme. We also note that two other datasets, **Congressional Voting Records** and **Australian**, perform similarly.

Figure 3.3 shows the results of running very similar advice (again $p = 0.75$) on the **Spambase** dataset. Here we can see that algorithms that strictly follow the advice outperform algorithms that ignore it, quite significantly. It is perhaps unsurprising that when we allow the algorithm more and more freedom to ignore the advice we move toward the baseline no-advice score. This highlights the fact that these algorithms are not always of use—there needs to be enough inaccuracy in the advice that attempting to follow it is not a great idea.

However, if we lower p to be 0.65, the situation changes, as demonstrated by Figure 3.4. Here as for the **Heart Disease** case, using only the Min-2-CC solution is worse than using no advice at all, and for a large range of f values, the compromise of using some advice is better than either extreme. The **Haberman** dataset performs similarly; the difference in this case is that for high f values, very poor performance is exhibited (see below).

Finally, we consider the **Hepatitis** dataset (Figure 3.5). Here we see new behaviour, as our algorithms only begin to perform well for high f values. Note as well that for this dataset, no algorithm performs well—a naive algorithm which clusters

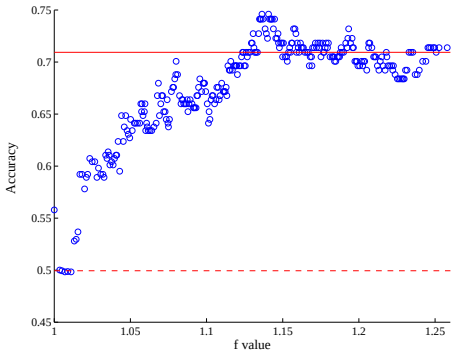


Figure 3.6: **Heart Disease** dataset, Complete advice, $p = 0.53$

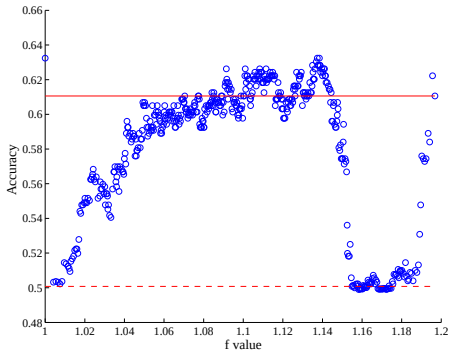


Figure 3.7: **Spambase** dataset, Complete advice, $p = 0.53$

all points in a single cluster will greatly outperform all of our algorithms. It seems that this problem is not a good candidate for a constrained clustering algorithm.

Complete Advice Figures 3.6 and 3.7 show the results of the experiments on the two datasets with Complete advice. We first notice that in order to get meaningful experiments, we needed to set p extra-ordinarily low—all the way down to $p = 0.53$. If p is much higher than this, advice is sufficiently strong that incorrect edges will be vastly overshadowed by correct ones, and simply solving Min-2-CC on the instance will give 100% accuracy.

However, with $p = 0.53$ (and the problem interesting), we can see that the situation is similar to the Dense case. Again, when f is low, we start at the

MIN-2-CC-baseline, and as f increases we move towards and above the no-advice baseline. An interesting point is that the MIN-2-CC baseline is around 0.5 in both experiments, which is an extremely low score. The advice alone is useless for solving the problem, yet it is still a useful addendum for the spectral method.

As we saw in the DENSE case, one interesting difference between the two datasets is the way the performance drops off as f increases. For **HEART DISEASE**, the performance seems to asymptote to the no-advice case as we increase f (as we would expect). However, in each case for the **SPAMBASE** data, there is a huge drop-off in performance for high-end f values. We have no explanation for this phenomenon.

3.8 Conclusion

In this chapter, we have seen a new algorithm that uses inconsistent advice in spectral clustering. The technique of transforming advice from a MIN-2-CC problem into a series of relaxed subspaces to constrain a SPECTRAL CLUSTERING or SDP algorithm was justified in terms of the degree to which we believe inconsistent advice. The algorithm that we generate was—in the spectral case—efficient, and as we have seen, more effective than approaches that use either the advice or the affinity data alone.

Chapter 4

Local Search

4.1 Introduction

In this chapter we will explore the intersection of the theoretical and practical approaches to solving graph optimisation problems. We will apply these ideas to the Min Feedback Arc Set (Min-FAS) and Min 2-Correlation Clustering (Min-2-CC) problems. These two problems share many characteristics, and can be approached in a somewhat similar fashion. In fact, we will find in this chapter that the best approach to take when solving each problem is local-search. This is a algorithmic technique based on repeatedly improving a solution until no further improvement is possible.

Apart from the obvious similarly between Min-FAS and Min-2-CC as simple graph-based minimisation problems, there are further connections that we will exploit in this chapter. Each problem has been studied in their its right for many years; a variety of algorithms have been introduced, especially in the case of Min-FAS. This problem in particular has seen a significant amount of practical investigation, resulting in some interesting heuristic algorithms.

Although we will have most success with relatively simple local-search algorithms for both problems, the problems share mathematical properties that inspire an interesting set of algorithms. This is due to the fact that the difficulties involved in both problems are related to *cycles* within the graphs. This insight leads us to the development of *cycle-destroying* algorithms for both problems, a new type of

algorithm, with connections to local-search techniques.

4.1.1 Local Search

Often, after running (our favourite) algorithm for a problem, we may be presented with a solution, which, after some small minor modification, can be improved slightly. For instance, a vertex may be swapped between clusters, or a pair of vertices could have their positions in the ranking changed, either of which could lead to a slightly improved solution to their respective problems.

Such an improvement is often called a *local improvement* as it doesn't involve changing anything fundamental or global about the problem, just making some local modification. By repeatedly applying this idea, we obtain a *local-search* algorithm, starting from some given (often random) solution, it continually makes such local improvements, until no further improvement is possible. Let us be precise.

Algorithm: Local Search (minimisation)

Given some neighbourhood function over the solutions to a problem $\mathcal{P}$:

$$\mathcal{N} : I \times \mathcal{S}(I) \to \text{Pow}(\mathcal{S}(I))$$

(Here $\mathcal{N}(I, S) \subseteq \mathcal{S}(I)$ is a set of solutions which are 'neighbours' of S).
Then we can define a Local Search algorithm as follows:

For an instance I, beginning at some solution $S_0 \in \mathcal{S}(I)$, perform the following procedure:

1. Let $S \leftarrow S_0$.
2. Choose an $S' \in \mathcal{N}(I, S)$ that minimises $\mathcal{P}(I, S')$.
3. If $\mathcal{P}(I, S') < \mathcal{P}(I, S)$, let $S \leftarrow S'$, go to step 2.
4. Otherwise return S.

The selection of S' in step 2, and the check made in step 3, as defined here, are very specific. We could instead make much more complicated decisions and checks, leading to the entire field of *metaheuristics*, such as Simulated Annealing and Tabu

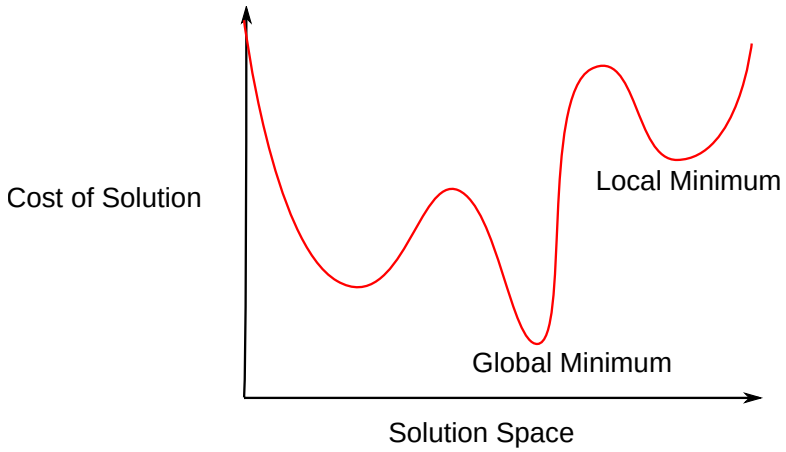


Figure 4.1: A diagram illustrating the problem with LOCAL SEARCH algorithms.

Search. However, in this thesis, for our local-search algorithms, we will always choose in step 2 a neighbour which minimises the objective function. Additionally, we will only change our solution if it *strictly improves* on the current solution. This approach is commonly known as *hill climbing*, but henceforth we will refer to it simply as LOCAL SEARCH.

Obviously, the behaviour of the algorithm very much depends on the choice of $\mathcal{N}$ and S^0. Usually we let S^0 be a random configuration. However, the main deficiency with every local-search algorithm is the same, no matter what choices of $\mathcal{N}$ is made: being stuck in a local optimum.

Figure 4.1 represents a simplified local-search algorithm, where the neighbourhood function $\mathcal{N}$ presents the algorithm with two choices ('move left' or 'move right'). We can see that, as the local-search algorithm moves the solution left and right, it is possible that it could get stuck in the local minimum on the right hand side. This is a solution of far greater cost than the global minimum. However, as the local-search algorithm is only looking in a small, local area around the current solution, it is not able to notice that there are far better solutions available.

For this reason, it is often very hard to provide guarantees about the behaviour of local-search algorithms. However, there have been some examples of proven results for LOCAL SEARCH algorithms. Arya et al. [8] were able to show that a simple swapping algorithm (the same as our TOSSES algorithm of Section 4.3.1) is a 5-approximation for the k-median problem; if in fact we allow up to p simultaneous

moves, a $3 + 2/p$-approximation is obtained. Kanungo et al. [64] demonstrate a similar algorithm is a 9-approximation for the k-means problem. Approximation results for such simple algorithms are very encouraging. However, we will see that the Tosses algorithm for Min-2-CC (and similar algorithms for Min-FAS) is not a constant-factor approximation algorithm.

Local-search algorithms are often used in practice, as they are often very fast and sometimes quite effective. Additionally, with random starting positions, they can be run multiple times to improve on the quality of results. More interesting is the possibility of using a local-search algorithm as an addendum to another, more complicated algorithm. Such a combination will have the approximation guarantees of the original algorithm, along with the practical performance of the local-search algorithm (although if the theoretical algorithm has a high running time, we do not gain the local-search algorithm's small running time).

In the rare case that a local-search algorithm has an approximation guarantee, we can add it as an addendum to a practical algorithm to achieve a performance guarantee, without adversely affecting the performance (and in most cases the running time).

4.1.2 Problem Instances

As we have seen in Section 2.3.4, the Min-FAS problem has been studied in the context of a number of applications. For these applications, the problem instantiates as many different types of digraphs. Similarly, practical applications of Min-2-CC generate a wide variety of signed graphs.

However, as outlined in Section 2.3.4, there as has been a flurry of activity in the approximation algorithms community recently with regards to tournament graphs (complete digraphs). The reason for this narrower focus is twofold. Firstly, the more regular, complete structure of a tournament encourages mathematical proofs. Secondly, one major class of applications of Min-FAS—those that are based on Rank Aggregation problems—generate tournament graph instances.

As we saw in Section 2.2.4, Min-2-CC has received limited attention from the computer science community, despite marked similarities to the Min Feedback Arc Set and Max-Cut problems. However, the few theoretical results that do

exist are also restricted to complete signed graphs, for similar reasons.

In this chapter, we are interested in marrying the work done in both spheres of endeavour, and trying to find which algorithms are truly 'best' for this problem. For this reason, we will mainly focus on complete graphs, as this is the type of problem for which most theoretical algorithms have been designed—it would hardly be a fair comparison otherwise.

4.1.3 Aims of this Chapter

The aim of this chapter is to discover which algorithms are most effective for the Min Feedback Arc Set and Min 2-Correlation Clustering problems, both from a practical as well as (where possible) a theoretical perspective.

In Section 4.2.1, we describe a series of algorithms that we develop for the Min-FAS problem, beginning with some very simple examples of Local Search algorithms, leading up to complex algorithms based on the idea of destroying directed triangles in tournament graphs. In Section 4.2.2 we will experimentally compare those algorithms with the existing algorithms for the Min-FAS problem, as described in Section 2.3.4. We conduct a thorough investigation, using both synthetic and non-synthetic datasets, in order to firmly establish the practicality of each algorithm. We complement that experimental work with some counter-examples demonstrating that many of the algorithms mentioned are not constant-factor approximation algorithms.

In Section 4.3.1, we describe another set of algorithms, for the Min-2-CC problems, with similarities to the algorithms described above. In addition to examples of Local Search, and a cycle-destroying algorithm (inspired by the directed triangle destroying algorithm), we also describe an algorithm, PASTA-toss, that uses the local-search step in combination with an algorithm improving on the Pick-a-Vertex algorithm of Bansal et al. [12]. Again, in Section 4.3.2, we perform an experimental investigation into which algorithm is practically most effective, on both synthetic and real-world datasets.

Finally, in Section 4.3.3, we provide a proof that the PASTA-toss algorithm is a 2-approximation on complete graphs, along with counter-examples that show its constituent algorithms are not constant-factor approximators.

4.2 Min Feedback Arc Set

4.2.1 The Algorithms

Local-Search Algorithms

As we mentioned in Section 4.1.1, local-search is a well know heuristic to solve optimisation problems. To define a local-search algorithm, we need to define the *neighbourhood* relationship—a method of generating neighbouring solutions from any specific potential solution. We can then make a choice of which of those neighbours to move to in each step of the Local Search algorithm.

The neighbourhoods we will consider in this thesis are very simple. In the case of Min-FAS, they involve making one simple improvement to a solution. We define two neighbour functions, which lead to two Local Search algorithms.

- The Swaps heuristic:

  > The neighbourhood $\mathcal{N}(\pi)$ of a ranking π is all rankings which can be obtained from π by swapping the position of two vertices.

- The Moves heuristic:

  > The neighbourhood $\mathcal{N}(\pi)$ of a ranking π is all rankings which can be obtained from π by moving one vertex to another position, leaving the relative order of all other vertices unchanged.

The Swaps algorithm runs in worst-case time $O(n^3)$ and the Moves algorithm in $O(n^4)$, although on average (as most local search algorithms do), perform much better. The two steps are demonstrated in Figure 4.2. We will show that neither algorithm can provide an approximation guarantee. On preliminary tests, we found that the Moves heuristic performed well, but that Swaps did not; the latter was omitted from further experiments.

The Chanas algorithm An application of the SORT step of the Chanas algorithm [19] (see Section 2.3.4) has the effect of checking, for each vertex of the graph, from left to right, if a Moves-style step, where the move is to the left, is possible.

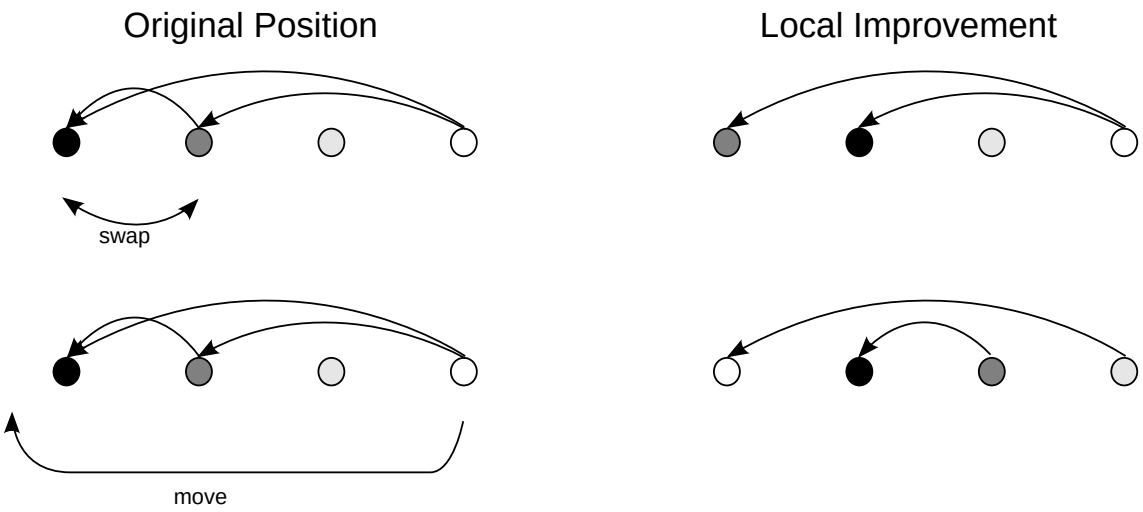


Figure 4.2: The two local-search steps that we consider in this chapter.

This is essentially a scheme for selecting which MOVES-style changes to make, as opposed to the gradient descent approach that we had decided upon. The operation SORT ∘ REVERSE does the same thing, but with moves to the right. So CHANAS is simply a method for investigating MOVES in a particular order. We developed an variant of this, which we call CHANAS BOTH, and which runs in similar time ($O(n^4)$):

Algorithm: CHANAS BOTH

> Run CHANAS, but change the SORT procedure, so we are allowed to move a node *either* left or right, to the position that results in the fewest back-arcs.

A consequence of this modification is that some nodes may be moved more than once in a single SORT pass.

Note that the BUBBLESORT algorithm, also described in Section 2.3.4, is a variant of SWAPS. It considers swapping adjacent vertices in a very specific order.

Triangle-Destroying Algorithms

A mentioned in Section 2.1, a tournament has a cycle if and only if it has a directed triangle (which we denote by $\vec{\triangle}$). This is due to the complete nature of a tournament

graph—every (non-$\vec{\triangle}$) cycle must have a chord inside it that forms a strictly smaller cycle. (see Figure 2.6). This of course means that if we can remove all the $\vec{\triangle}$s from a tournament, we will have an acyclic tournament, which can be ranked easily.

We therefore consider algorithms that destroy directed triangles by selecting arcs to reverse. It might seem more natural to delete arcs, but this would make the digraph no longer a tournament, creating the possibility of cycles without the presence of $\vec{\triangle}$s. Our scheme works in the following way:

Algorithm: Triangle Deletion

> While the tournament is not acyclic, *choose* an arc and reverse its orientation. Once the tournament is acyclic, use the Topological Sort of the vertices as the solution to the original problem.

The choice of arc to be reversed affects the performance and running time of this procedure; the remainder of this section examines various heuristics. Figure 4.3 demonstrates the operation of a Triangle Deletion algorithm.

We call the number of $\vec{\triangle}$s an arc is involved in its *triangle count*. The triangle count of a tournament is simply the number of $\vec{\triangle}$s in that tournament—as each $\vec{\triangle}$ contains three arcs, the tournament has a triangle count equal to the one-third of the collective sum of the triangle count of its arcs. Our first algorithm is:

Algorithm: Triangle Count

> Run Triangle Deletion, choosing on each iteration the arc with highest triangle count.

There is a pitfall here though: reversing an arc can create a new $\vec{\triangle}$ that did not previously exist. In Figure 4.4 we see that reversing the center arc does not actually reduce the triangle count of the graph, as a single triangle is destroyed and new triangle is created. The problem here is really the length-4 cycle surrounding the arc, and the solution is to reverse one of these outer arcs (one of which in fact has triangle count two). So the worst case running time of Triangle Count is unbounded.

However there are (more complex) examples where reversing the arc with the highest triangle count creates the same number of triangles as it destroys. So that

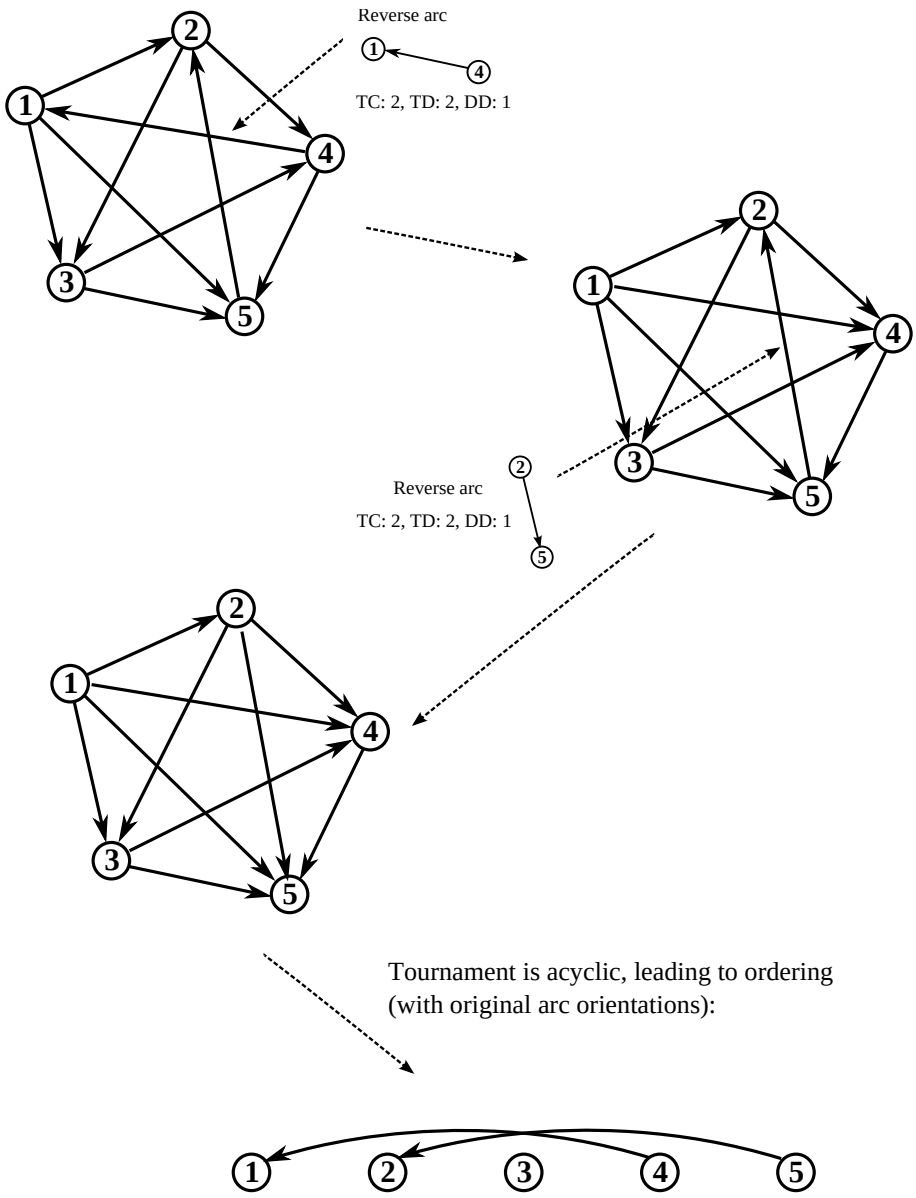


Figure 4.3: An example of an TRIANGLE DELETION algorithm in action. TC refers to the triangle count, TD to the triangle delta, and DD to the degree difference, as explained below. The arcs chosen could be chosen by any of our TRIANGLE DELETION algorithms. We re-arrange the tournament so that it becomes acyclic; we can then order it easily, however the arcs that we have reversed are now back-arcs.

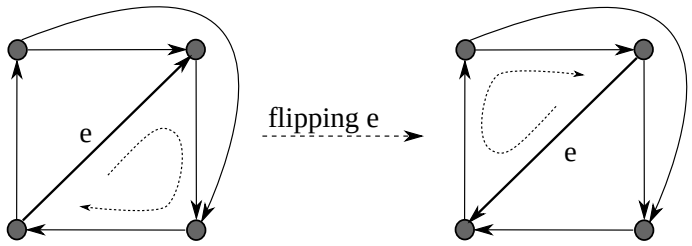


Figure 4.4: An example where reversing creates a triangle whilst destroying another.

arc still has the highest triangle count, leading to an infinite loop.

To avoid this problem, we define the following algorithm:

Algorithm: Triangle Delta

> Run Triangle Deletion, choosing the arc which causes the greatest net reduction to the tournament's triangle count.

A potential problem with Triangle Delta could be the existence of a tournament that was not acyclic (and thus still had $\vec{\triangle}$s, and thus a positive triangle count), yet contained no arcs whose reversal would lower the triangle count. Lemma 1 proves that this situation is impossible.

Lemma 1. *Let T be a tournament. If T has a cycle, then there exists an arc $e \in T$ such that reversing e will reduce the triangle count of T.*

Proof. Let σ be an ordering of the vertices of T that induces a minimum feedback arc set. Let $a = v \leftarrow w$ be a back-arc of maximal length under σ, that is maximising $\sigma(w) - \sigma(v)$. We claim that reversing a will lower the triangle count.

Firstly, we note that reversing a will not create any $\vec{\triangle}$s of the form v-w-x, where x is to the right of both v and w, as this would imply a back-arc $v \leftarrow x$ that is 'longer' than $v \leftarrow w$; this is impossible by our choice of a. Similarly, no $\vec{\triangle}$ x-v-w can be created where x is to the left of both vertices. So any $\vec{\triangle}$ created must involve an x between v and w.

Consider the four possibilities for a node x that is placed between v and w by σ:

1. $v \overset{a}{\leftarrow} x \leftarrow w$ (reversing a will create a $\vec{\triangle}$). Say there are A such xs.
2. $v \overset{a}{\leftarrow} x \rightarrow w$ (reversing a will delete a $\vec{\triangle}$). B of these.

3. $v \xleftarrow{a} x \longleftarrow w$ (reversing a will have no effect). C of these.

4. $v \xleftarrow{a} x \longrightarrow w$ (no effect). D of these.

Since σ is optimal, moving v to the position after w will not reduce the back-arc count. So the number of back-arcs into v from such x's must be less than the number of forward arcs from v to such x's (strictly, as there is a back-arc from w to v). So we have

$$A + D < B + C\,.$$

Similarly, moving w before v will not improve the order, so we have

$$A + C < B + D\,.$$

Combining these gives

$$2A + D + C < 2B + C + D \implies A < B\,,$$

and therefore the number of $\vec{\triangle}$s will decrease. □

As there are up-to $O(n^3)$ triangles on a graph, a TRIANGLE DELTA algorithm will take time $O(n^5)$, although, like most local search algorithms, will tend to do much better, as many triangles will be deleted in each step.

In practice, contrary to our expectations, the TRIANGLE COUNT algorithm tended to outperform the TRIANGLE DELTA algorithm (except of course in the cases when it did not complete execution). So we considered a third option, taking the best of both approaches, again running in $O(n^5)$:

Algorithm: TRIANGLE BOTH

> Run TRIANGLE DELETION, choosing the arc with the highest triangle count, provided that it reduces the tournament's total number of $\vec{\triangle}$s.

Note that the best algorithm we have for calculating the triangle count, and the change in $\vec{\triangle}$s, for every arc of the digraph requires $O(n^3)$ operations. It seems unlikely that this can be improved as there are potentially $O(n^3)$ such $\vec{\triangle}$s.

In a weighted tournament, the weight of a $\vec{\triangle}$ is the sum of the weights of its arcs. Therefore in such graphs, the triangle count of an arc is the sum of the weights of the $\vec{\triangle}$s it is involved in.

Degree Difference Algorithms

We designed a new algorithm, Degree Difference, which is again a triangle-deletion algorithm, but which selects an arc to reverse based on a criterion that is much simpler to compute than the full triangle count. However, the criterion seems empirically to be related to the triangle count.

Algorithm: Degree Difference

> Run Triangle Deletion, choosing the arc $u \to v$ for which the difference between u's indegree and v's indegree is greatest.

Unfortunately, it may take $\Theta(n)$ time to find such an arc at each iteration. Nevertheless, as we now show, this algorithm always makes progress towards a total ordering. Whilst the tournament is non-transitive, there must be an arc with non-negative degree difference. If there were no such arc, the solution returned by Iterated Kendall would have no back arcs, which is a contradiction, as a perfect solution implies the digraph is transitive. But the value of $\sum_v \mathrm{In}(v)^2$ increases whenever an arc of non-negative degree difference is reversed, and has a maximum value of $\sum_{i=0}^{n-1} i^2$ when the tournament has a total ordering. This means the total number of steps is $O(n^2)$, leading to a $O(n^3)$ algorithm. However, this may still lead to a time-consuming algorithm, due to the time taken to find the arc of highest degree difference.

In an effort to further speed up the Degree Difference algorithm, we used a sampling technique. We sample $\log n$ vertices (favouring high indegree) to potentially be the 'tail' of the arc, and another $\log n$ (favouring low indegree) to potentially be the 'head'. We then check each of the $\log^2 n$ arcs between sampled vertices, choosing the back-arc of highest degree difference. We resample if we find back-arcs only of non-positive degree difference. This algorithm is called Degree Difference Sampled 1, and it takes $O(n^2 \log^2 n)$ time on average.

A further variation, Degree Difference Sampled 2, maintains two lists: one

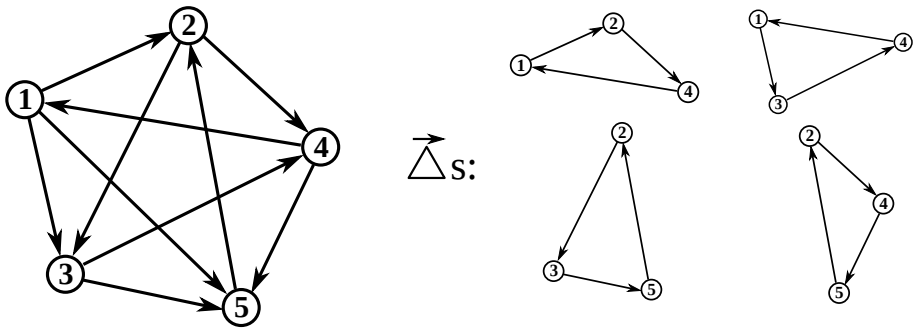


Figure 4.5: An example of the calculations of the various TRIANGLE DELETION algorithms. This tournament contains four directed triangles. The arcs $4 \to 1$ and $5 \to 2$ have triangle count and triangle delta scores of two, the highest on the tournament. Additionally, they have degree difference 1. Vertices 4 and 5, with high indegree, are good candidates for tails, likewise 1 and 2 are good head candidates. Each algorithm will consider them good arcs to reverse. Note that for instance the arc $1 \to 5$ has triangle count zero, and triangle delta -1, due to the potential $\vec{\triangle}$ $1 \to 3 \to 5 \to 1$, and has degree difference -2. This is a poor arc to reverse.

of potential head nodes, and one of potential tail nodes. The idea is to try to push the quantity $\sum_v \mathrm{In}(v)^2$ towards its maximum—which is reached when all indegrees are different, and the graph is thus transitive. With this in mind, remembering that reversing an arc will increase the indegree of it's tail and decrease the indegree of its head, a node v is a potential head if $\mathrm{In}(v)$ is not unique or there is no node of indegree $\mathrm{In}(v) - 1$. Similarly, a node u is a potential tail if $\mathrm{In}(u)$ is not unique or there is no node of indegree $\mathrm{In}(u) + 1$. We maintain lists of both potentiates, and sample $\log n$ nodes from each list uniformly, selecting for the $\log^2 n$ pairs the arc with the largest degree difference to reverse. Again, this algorithm will take time $O(n^2 \log n)$.

4.2.2 Experiments

We conducted a series of experiments to validate the empirical performance of these algorithms. All experiments were conducted on a 4-core Intel Xeon 3.2GHz machine, with 8 gigabytes of physical memory. All algorithms were compiled by `gcc` version 3.4.6 with the `-O3` optimisation flag. Note that in all relevant algorithms we broke ties randomly.

Algorithms Tested

Apart from the new algorithms listed above, we also tested the following algorithms, as described in Section 2.3:

- The Iterated Kendall [38] algorithm (described on Page 62).
- The Eades [39] algorithm (analogous to Selectionsort, Page 63).
- The Chanas [19] algorithm (described on Page 65).
- Our modification Chanas Both, as described above.
- Ailon et al.'s [2] Quicksort algorithm (described on Page 68), along with the obvious generalisation to Bubblesort and Mergesort.

Note that the PTAS of Kenyon and Schudy [69] was unfortunately too complicated and impractical to implement.

Eades Improved The Eades algorithm focuses on the left side of the ranking. We improve this by allowing the selection of a vertex to either end of the ranking. For instance, if there is a vertex v of extremely low outdegree, but no vertex of low indegree, it makes sense to decide to place v on the right hand side of the ordering, deferring the more difficult choice on the left.

Remember $\text{In}(v)$ is the indegree of node v, and $\text{Out}(v)$ its outdegree. Our improved algorithm is (running, as Eades in time $O(n^2)$):

Algorithm: EADES IMPROVED

- Choose the vertex m that maximises $\big|\mathrm{In}(u) - \mathrm{Out}(u)\big|$.
- Let $\pi_0 = \textsc{Eades}(G|_{V\setminus\{m\}})$.
- If $\mathrm{In}(m) < \mathrm{Out}(m)$, return

$$\pi(v) = \begin{cases} 1 & \text{if } v = m, \\ \pi_0(v) + 1 & \text{otherwise.} \end{cases}$$

- Otherwise, return

$$\pi(v) = \begin{cases} n & \text{if } v = m, \\ \pi_0(v) & \text{otherwise.} \end{cases}$$

In order to investigate the significance of initial solution quality to the effectiveness of local-search techniques, we first tested each algorithm in isolation—for the local-search algorithms, this meant starting from a random ordering. Following this, we then passed the output of each algorithm into both the CHANAS and MOVES algorithms.

Note that passing the output of CHANAS as input to CHANAS is a surprising case. CHANAS is a local-search algorithm—which would imply that as the output of CHANAS is locally optimal, a further call to CHANAS would have no further effect. However, a single call to the SORT* $\circ$ REVERSE step can significantly change the ordering without affecting the solution quality. That is, whilst deciding that it is at a local minimum, the algorithm can take a "sideways step" (one which changes the ordering, without decreasing the cost). So it is possible that two calls to SORT* $\circ$ REVERSE can lower the cost of a ranking, whilst one will not. Repeated calls to CHANAS can sometimes move the algorithm out of a local plateau. For this reason, CHANAS + CHANAS can sometimes improve on, and also take longer than, CHANAS alone. However, it is difficult to predict when this is going to happen, and so no systematic method to take advantage of this is apparent.

Datasets

Biased We tested the Min-FAS algorithms on the following synthetic dataset. Starting with a total order from nodes 1 to 100, we reverse each arc independently with probability p. In particular, with $p = 0.5$, we have a completely random tournament.

The following datasets provide a set of rankings to be aggregated—the Rank Aggregation problem on these rankings provides us with a Min-FAS tournament as described in Section 2.3.3.

WebCommunities Laurence Park provided us with a set of 9 rankings of a large set of documents (25 million) [89]. From this we took 50 samples of 100 documents and considered the rankings of each of those subsets.

EachMovie We used the EachMovie collaborative filtering dataset [79] to generate tournaments of movie rankings. The idea here was to identify subgroups (we used simple age/sex demographics) of the users, and then generate tournaments that represented the 'consensus view' of those groups.

The EachMovie dataset consists of a vote (on a scale of one to five) by each user for some set of the movies. To form a tournament from a group we took the union of movies voted for by that group and then set the arc weight from movie a to movie b to be the proportion of users who voted a higher than b. For consistency with the other datasets, we sampled each tournament down to size 100.

Discussion of Results

We tested all of the algorithms on a large number of data sets. We selected just four of the data sets to display in Table 4.1: these show a variety of performance characteristics. Listed for each variant is the percentage error, as compared to our baseline algorithm—Chanas in isolation. Also listed is the percentage of 'wins'—that is the number of times the algorithm does the best (of the variant). If k algorithms each produce the best solution on a given input graph, each is given $1/k$ wins. Finally, the total time taken, in seconds, is given.

		0.6			0.95			WEBCOMMUNITIES			EACHMOVIE		
	Variant	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**	**Errors**	**Wins**	**Time**
	—	12.02	0.0	2.1	29.79	0.0	1.9	15.63	0.0	0.1	8.39	0.0	0.3
IT. KEND.	MOVE	0.35	7.5	4.9	0.30	10.0	3.8	0.00	12.0	0.4	0.31	7.4	0.8
	CHAN	-0.35	10.9	6.7	0.00	14.5	4.5	0.00	13.6	0.3	-0.06	7.8	1.0
	—	9.88	0.0	4.7	62.02	0.0	4.7	31.38	0.0	0.2	7.47	0.0	0.7
EADES	MOVE	0.56	4.0	7.5	0.31	5.5	6.9	0.01	5.3	0.6	0.25	11.5	1.1
	CHAN	-0.29	9.6	9.7	0.00	7.3	7.3	-0.00	6.7	0.4	-0.08	8.0	1.4
	—	8.65	0.0	1.9	52.10	0.0	1.9	19.29	0.0	0.1	6.37	0.0	0.3
EADES IMP.	MOVE	0.60	3.2	4.7	0.32	5.2	4.7	0.00	8.6	0.3	0.37	2.2	0.8
	CHAN	-0.20	6.9	6.8	0.00	7.3	4.7	0.00	7.0	0.3	-0.02	6.3	1.0
	—	0.00	74.0	5.6	0.00	27.8	2.8	0.00	23.6	0.1	0.00	83.8	0.7
CHANAS	MOVE	-0.01	34.1	7.5	-0.00	12.8	5.3	-0.00	9.4	0.3	-0.03	54.5	1.1
	CHAN	-0.04	6.4	7.8	-0.00	7.3	5.5	-0.00	7.0	0.3	-0.04	9.7	1.1
	—	35.25	0.0	1.6	728.46	0.0	1.6	74.46	0.0	0.1	31.07	0.0	0.3
BUBBLE.	MOVE	0.75	4.0	4.6	0.21	7.0	3.4	0.00	5.4	0.3	0.33	1.5	0.8
	CHAN	-0.02	4.4	7.2	0.00	7.2	3.6	-0.00	8.9	0.3	-0.01	7.8	1.0
	—	23.23	0.0	1.6	130.81	0.0	1.6	0.86	1.4	0.1	20.62	0.0	0.3
MERGE.	MOVE	0.79	2.7	4.6	0.23	6.4	5.2	0.00	6.0	0.2	0.34	3.4	0.8
	CHAN	-0.02	5.6	7.2	0.00	7.2	4.5	0.00	6.6	0.3	-0.01	7.4	1.1
	—	23.65	0.0	1.6	135.63	0.0	1.6	0.91	1.5	0.1	20.27	0.0	0.3
QUICK.	MOVE	0.78	3.7	4.6	0.22	6.9	4.4	0.00	6.3	0.2	0.34	1.7	0.8
	CHAN	-0.01	4.9	7.1	0.01	7.2	4.5	0.00	6.5	0.2	0.01	8.1	1.0
	—	2.12	0.2	345.4	0.06	21.6	208.9	0.01	19.3	2.7	4.42	0.0	52.5
TRI. BOTH	MOVE	0.23	12.4	347.9	0.05	10.4	211.1	0.00	8.7	2.8	0.26	5.5	52.9
	CHAN	-0.40	13.4	349.8	0.03	6.8	211.8	-0.00	7.0	2.8	-0.07	10.1	53.2
	—	11.24	0.0	9.4	48.05	0.0	4.0	0.42	0.0	0.1	9.65	0.0	1.3
D. D. SAM. 1	MOVE	0.30	9.6	12.2	0.29	5.6	5.8	0.00	7.1	0.3	0.29	2.2	1.8
	CHAN	-0.40	11.8	14.0	0.00	7.2	6.1	-0.00	7.5	0.3	-0.05	6.6	2.0
	—	10.75	0.0	15.6	99.82	0.0	6.0	0.86	0.0	0.1	9.38	0.0	2.4
D. D. SAM. 2	MOVE	0.33	8.1	18.3	0.27	5.9	8.4	0.00	6.2	0.3	0.32	4.4	2.8
	CHAN	-0.37	11.6	20.1	0.00	7.2	8.7	-0.00	7.1	0.3	-0.07	11.1	3.1
	—	0.85	11.8	17.6	0.28	12.0	11.5	0.00	25.1	0.7	0.35	8.1	2.6
MOVES	MOVE	0.85	2.2	19.4	0.28	5.7	14.3	0.00	8.8	0.8	0.35	2.9	2.9
	CHAN	-0.07	3.9	21.6	0.03	6.7	14.2	-0.00	7.2	0.8	-0.08	9.6	3.1
	—	0.77	14.0	3.0	0.21	14.3	3.6	0.00	18.5	0.2	0.31	8.1	0.5
CHAN. BOTH	MOVE	0.77	3.6	4.8	0.21	6.7	5.4	0.00	6.7	0.3	0.31	3.0	0.8
	CHAN	-0.09	5.1	7.0	0.03	6.7	5.9	-0.00	7.4	0.3	-0.05	7.5	1.0

Table 4.1: Algorithms were tested on the **BIASED** data set with $p = 0.6$ and 0.95, and on the **WEBCOMMUNITIES** and **EACHMOVIE** data sets. In addition to the basic algorithms, we ran MOVES and CHANAS-style local search procedures on the outcomes of each. We report the average of the *percentage* (%) relative difference between the number of back-edges (**Errors**), compared to the CHANAS heuristic, over all problem instances in the dataset. We also report the percentage (%) of times each algorithm wins (produces the best amongst the solutions generated), with the win split between algorithms that are equal first on a particular instances (**Wins**). **Time** is the average running time (in seconds) of each (combination of) procedure(s). Note that both **BIASED** datasets had 100 instances, **WEBCOMMUNITIES** 50, and **EACHMOVIE** 146.

We first note the striking performance of the Chanas local-search procedure, which is rarely beaten. Despite coming with an approximation guarantee, the Quicksort procedure performs relatively poorly, as does the Mergesort algorithm. Bubblesort does surprisingly well on the **WebCommunities** dataset, although it seems from the results that the **WebCommunities** dataset is a particularly simple problem instance. Also, Bubblesort only does well after the Chanas procedure is applied; otherwise it is possibly the worst of the algorithms. The Eades and Eades Improved algorithms are strong, but there should be a preference for the latter due to its lower running time.

As expected, the Triangle Both algorithm is very slow, though it is a strong performer when combined with local-search. The Degree Difference algorithm is similar, though at a different point on the effectiveness/speed trade-off. The sampling methods for Degree Difference—Degree Difference Sampled 1 and Degree Difference Sampled 2—seem better compromises.

The Time-Effectiveness Trade-off

Figures 4.6 to 4.9 highlights the trade-off between speed and efficiency of selected algorithms. On the Biased ($p = 0.6$) data set, the two algorithms that cannot be said to be worse than others (as they are on the *efficient frontier*) are the hybrid of Degree Difference Sampled 1 and Chanas, and the hybrid of Iterated Kendall and Chanas. Chanas by itself, not shown in this picture, unsurprisingly takes less time than these two algorithms. Chanas Both and Eades deserve special mention for their performance on the **WebCommunities** and **EachMovie** datasets respectively.

However, in the experiments to follow, we will consider only the **Biased** dataset. This is due to the ease of generating many problem instances of differing sizes for that synthetic dataset. For this reason, in what follows, we will focus our attention on the three algorithms which perform best on **Biased**: Chanas, Iterated Kendall and Degree Difference Sampled 1.

In Figures 4.10(a) and 4.10(b), we test the benefit of multiple runs of a randomised algorithm—to see if there is an effective sacrifice of running time in comparison to solution quality.

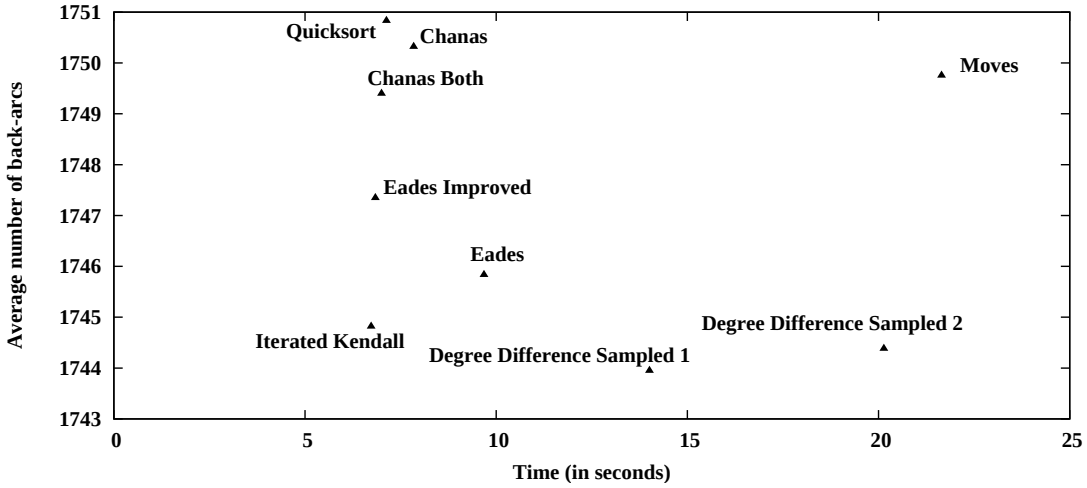


Figure 4.6: The trade-off between amount of time taken compared to the effectiveness of the various algorithms as inputs to CHANAS. A point to the left indicates reduced running time; further downwards indicates fewer errors in the output ranking. The data shown is from the **BIASED** dataset, $p = 0.6$, from a run of 1000 instances of size 100. On the efficient frontier (the most valuable algorithms) are those that have no other algorithm both below and to the left–these include ITERATED KENDALL, and DEGREE DIFFERENCE SAMPLED 1.

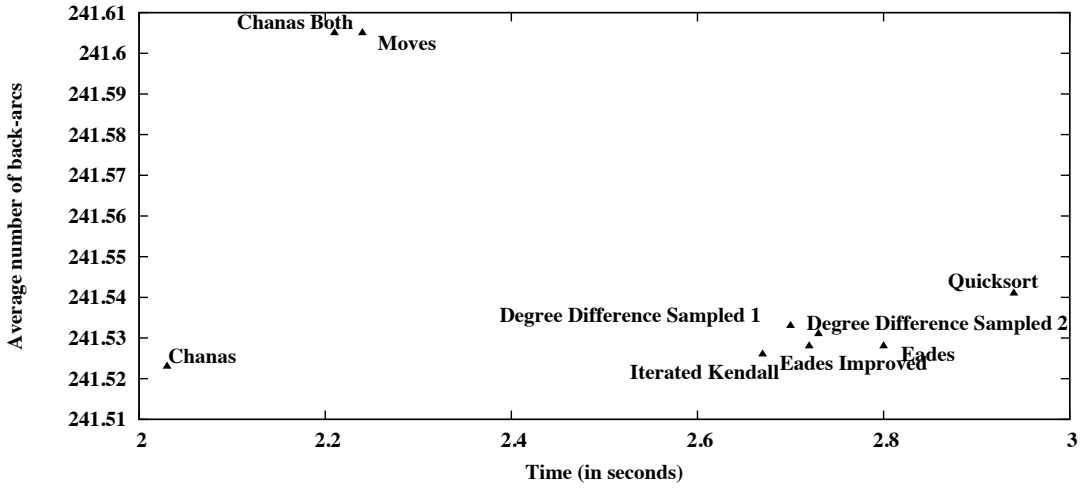


Figure 4.7: The same time versus effectiveness trade-off for the **BIASED** dataset, $p = 0.9$. Again a run of 1000 instances of size 100. Here CHANAS holds a clear advantage.

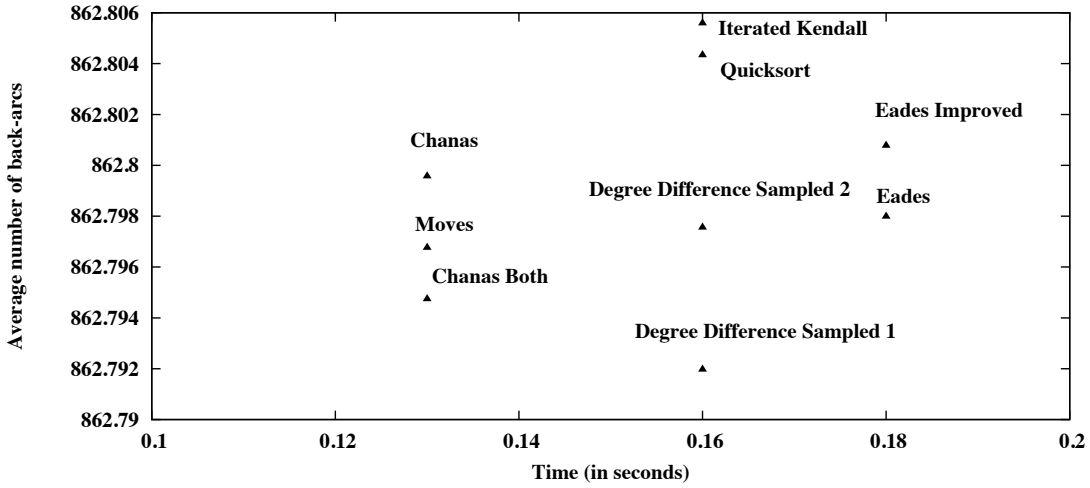


Figure 4.8: The same time versus effectiveness trade-off for the **WebCommunities**, over the 50 problem instances. For this dataset Degree Difference Sampled 1 and Chanas Both are on the frontier.

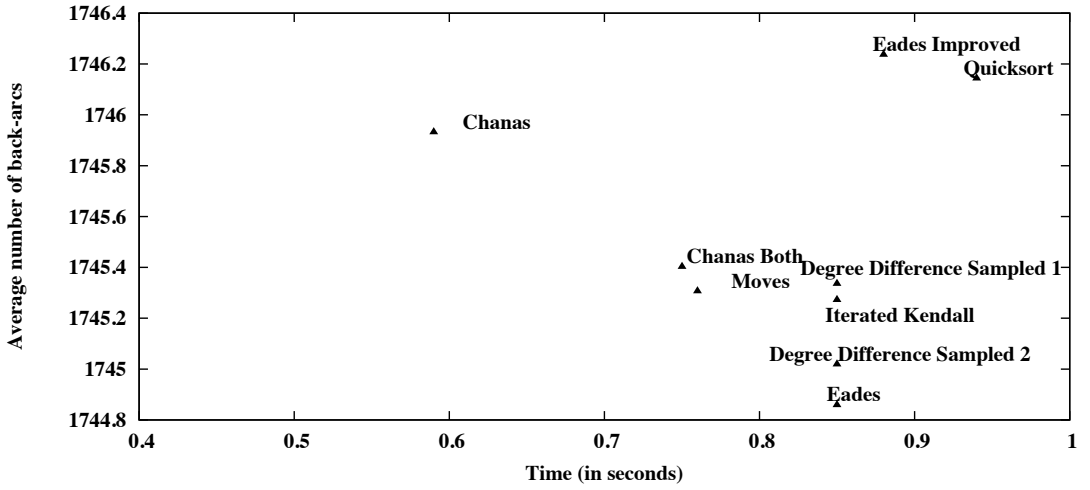
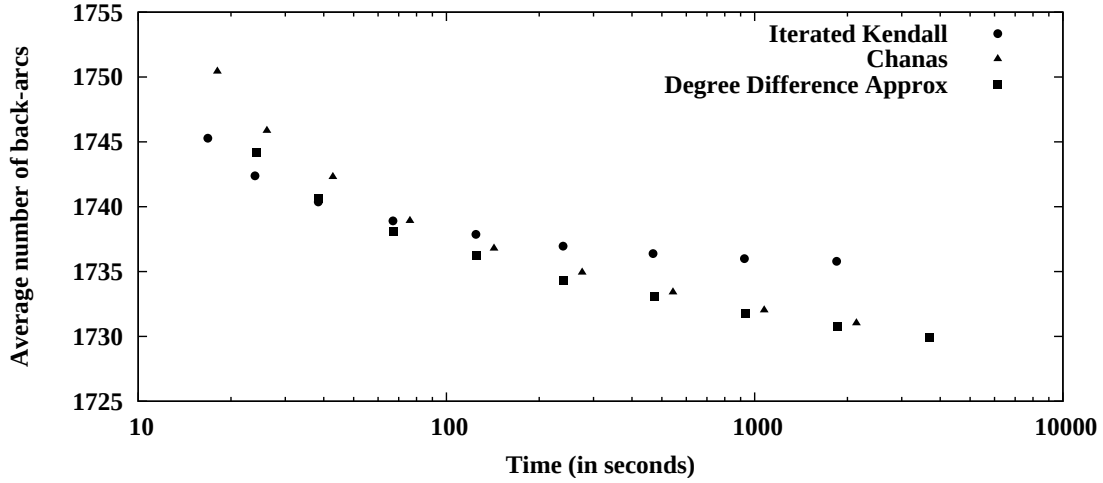
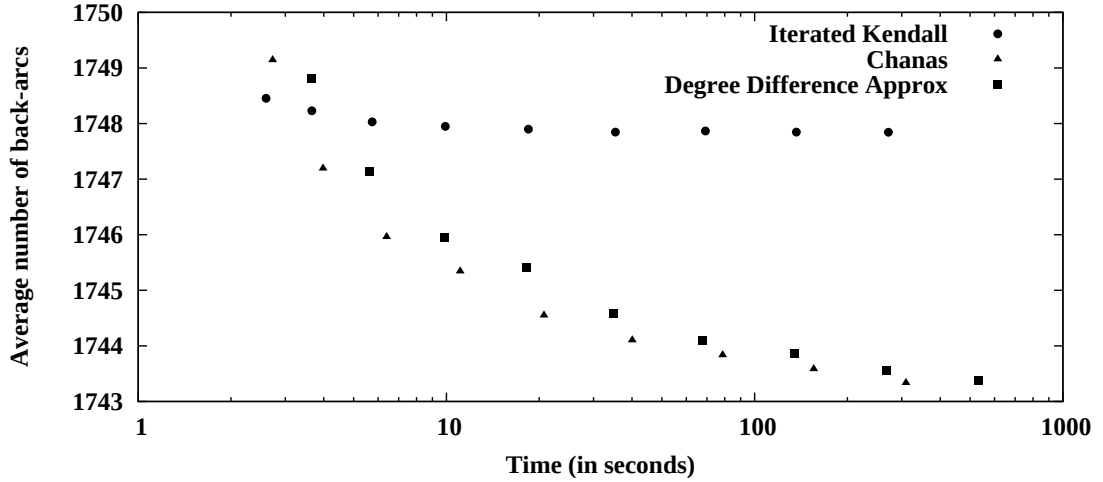


Figure 4.9: The same time versus effectiveness trade-off for the **EachMovie**, over the 146 problem instances. Chanas and Eades seem the more effective algorithms here.



(a) For the BIASED dataset, $p = 0.6$.



(b) For the EACHMOVIE dataset (146 tournaments of size up to 100)

Figure 4.10: The effect of repeated calls to a hybrid of each algorithm and CHANAS. We ran each algorithm once, twice, four times, etc. up to 256 times. Displayed for each run is the best-performing output over all repeats.

	Iterated Kendall				Chanas				Degree Difference Approx			
Rep.	**Time**	**Avg.**	**Best**	**Worst**	**Time**	**Avg.**	**Best**	**Worst**	**Time**	**Avg.**	**Best**	**Worst**
1	19.2	1745.11	1745.11	1745.11	20.3	1750.22	1750.22	1750.22	26.3	1744.65	1744.65	1744.65
2	26.5	1745.02	1742.36	1747.68	28.8	1750.54	1745.35	1755.73	40.8	1743.92	1740.38	1747.45
4	40.6	1744.90	1740.22	1749.93	45.2	1750.57	1741.65	1760.51	69.4	1744.23	1738.08	1750.61
8	69.2	1745.08	1738.76	1751.75	78.5	1750.66	1738.87	1764.53	127.0	1744.24	1735.99	1753.47
16	126.5	1745.04	1737.73	1753.11	144.9	1750.53	1736.44	1768.42	242.0	1744.25	1734.30	1755.78
32	240.7	1745.08	1736.76	1754.33	277.4	1750.57	1734.68	1772.00	471.2	1744.25	1732.88	1757.80
64	468.2	1745.06	1736.09	1755.02	542.2	1750.57	1732.97	1775.53	929.8	1744.25	1731.68	1760.13
128	924.3	1745.05	1735.78	1755.53	1071.6	1750.57	1731.80	1778.49	1848.4	1744.24	1730.60	1762.00
256	1837.6	1745.04	1735.48	1755.94	2133.4	1750.55	1730.67	1781.28	3689.8	1744.24	1729.69	1763.69

Table 4.2: The effect of repeated calls to a hybrid of each algorithm and Chanas. We ran each algorithm multiple times and recorded the total running time as well as the best, worst and average performance. These results are for the **Biased** $p = 0.6$ dataset.

	Iterated Kendall				Chanas				Degree Difference Approx			
Rep.	**Time**	**Avg.**	**Best**	**Worst**	**Time**	**Avg.**	**Best**	**Worst**	**Time**	**Avg.**	**Best**	**Worst**
1	3.0	1750.58	1750.58	1750.58	3.1	1750.68	1750.68	1750.68	3.9	1751.39	1751.39	1751.39
2	4.0	1750.60	1750.27	1750.93	4.3	1751.08	1749.53	1752.63	6.1	1750.77	1749.21	1752.34
4	6.1	1750.60	1750.11	1751.12	6.7	1750.82	1747.91	1754.04	10.2	1750.78	1748.13	1753.91
8	10.3	1750.54	1750.02	1751.21	11.7	1751.06	1747.26	1755.91	18.4	1750.78	1747.25	1755.26
16	18.6	1750.56	1749.91	1751.32	21.3	1751.11	1746.62	1758.06	35.0	1750.78	1746.77	1757.19
32	35.7	1750.58	1749.91	1751.38	40.5	1751.03	1746.28	1759.03	68.0	1750.74	1746.08	1758.13
64	69.6	1750.57	1749.88	1751.40	79.0	1751.07	1745.84	1761.29	134.1	1750.73	1745.84	1759.62
128	137.1	1750.59	1749.87	1751.41	156.7	1751.07	1745.59	1762.00	266.4	1750.74	1745.52	1760.30
256	272.8	1750.57	1749.87	1751.41	311.9	1751.09	1745.28	1763.47	531.8	1750.75	1745.31	1762.06

Table 4.3: Similar data to Table 4.2, for the **EachMovie** dataset.

In attempt to give each algorithm an equal amount of time to operate, we repeat each algorithm twice, four times, eight times, etc. On each run, we execute CHANAS as a finishing step. In Figures 4.10(a) and 4.10(b), we plot the best result found over the set of repeats, along with the total time taken, in seconds. In Tables 4.2 and 4.3 we display the numerical results, along with the average and worst result for each set of repeats.

There is a certain random component to all of these algorithms. For ITERATED KENDALL, there is less randomness in the algorithm, and this is borne out in the results. This leads to a relative stagnation in its effectiveness, especially on the **EACHMOVIE** data. The hybrid CHANAS and DEGREE DIFFERENCE SAMPLED 1 algorithms perform similarly on both datasets, however DEGREE DIFFERENCE SAMPLED 1 shows some advantage on the **BIASED** dataset, whilst CHANAS has a clear advantage on **EACHMOVIE**. Naturally, one could run Wilcoxon-style [106] non-parametric tests to show that one algorithm is significantly better than the other in a pure statistical sense. However, the difference may not be important, and it can be hard to compare algorithms that take slightly different running times. We leave the graphs themselves as the strongest evidence of the similar performance.

Increasing Problem Size

We investigated the running times of the algorithms on larger data sets to infer something about the running time growth (see Figure 4.11). In the diagram, we have separated the algorithms into seven classes, each with roughly similar running times (generally within 10% for each sample). We can see immediately that TRIANGLE BOTH and DEGREE DIFFERENCE have very high running times, corresponding to their cubic order of growth. The fastest class includes ITERATED KENDALL and QUICKSORT as well as CHANAS BOTH (but not CHANAS) and EADES IMPROVED (but not EADES), which appear to have quadratic growth. So our improved algorithms deliver a speed increase.

The middle range of speeds has CHANAS, and EADES, along with DEGREE DIFFERENCE SAMPLED 1 highlighting that DEGREE DIFFERENCE SAMPLED 1 runs in asymptotically similar time to CHANAS, although for larger data sets DEGREE DIFFERENCE SAMPLED 1 seems to have an advantage. The final class includes

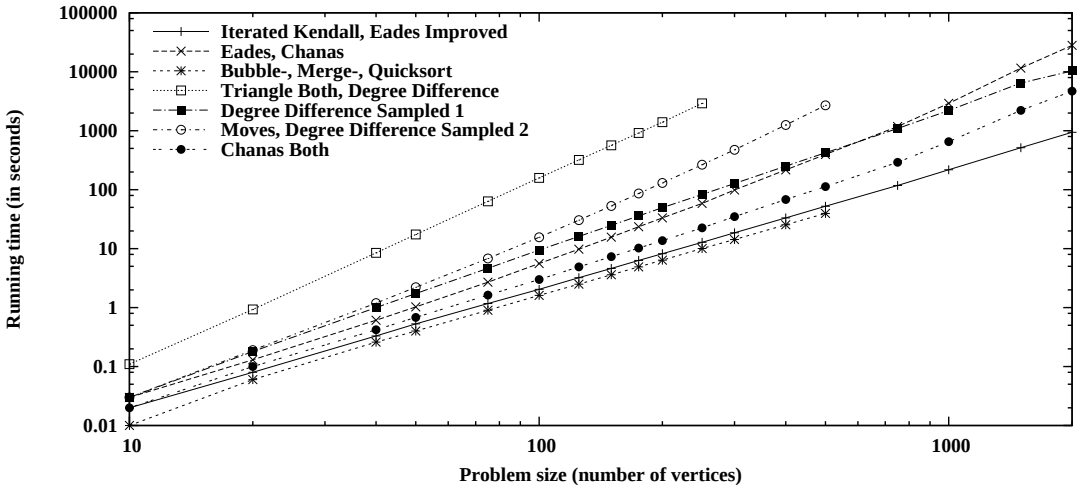


Figure 4.11: The running time taken by a selection of the algorithms, as the problem size increases. Each plot represents a class of algorithms with similar running times. All experiments were run on the **Biased** dataset, $p = 0.6$.

Degree Difference Sampled 2 and Moves, which also appear to have cubic growth.

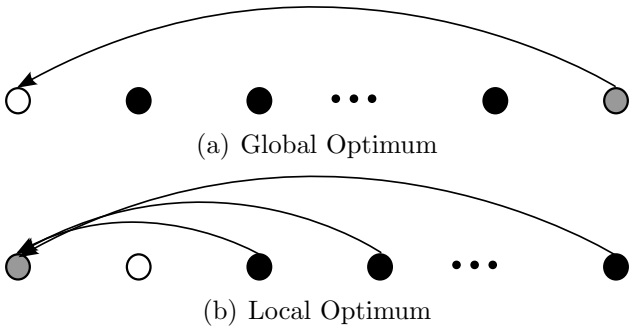
4.2.3 Theoretical Results

We now show that various algorithms for Min-FAS cannot guarantee constant-factor approximations. This is opposed to the previous section, where we experimentally tested each algorithm to try and find which is most practical.

All graphs shown are tournaments but in the interest of readability, sometimes not all arcs are shown. As in Chapter 2, in the Min-FAS figures below, only back-arcs, with respect to the given ordering, are displayed. All pairs of nodes with no arc displayed are assumed to have a right-pointing arc between them. We remind the reader that the cost of an ordering is simply the total number of back-arcs.

Standard Bad Example

This example consists of a completely transitive tournament of size n, with one minor perturbation—there is a single back-arc, from the last node (node n) to the first (node 1). This example can catch out algorithms which are "too local" in



(a) Global Optimum

(b) Local Optimum

Figure 4.12: Standard Bad Example: the global optimum has a single back-edge, whilst the local optimum (for the SWAPS heuristic) has $n-2$. The only difference between the two is in the placement of the grey vertex—all other nodes are unchanged.

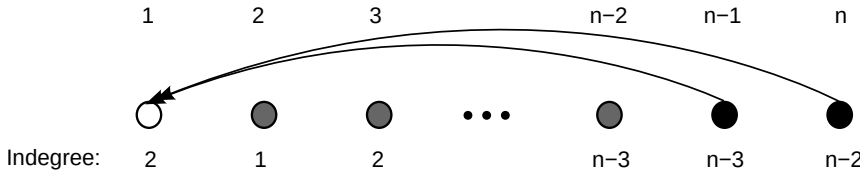


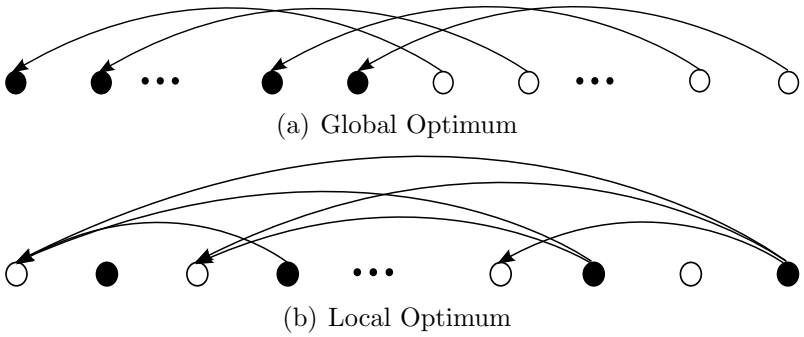
Figure 4.13: The Counterexample for the EADES algorithm. The global optimum is as pictured. The EADES algorithm will place the white node at the rightmost position, creating $n-3$ back-arcs, whilst removing only 2.

their operation. Figure 4.12(a) shows this (global) optimum configuration; a local optimum for the SWAPS heuristic is shown in Figure 4.12(b), with cost $n-2$.

Note also that there is no guarantee that BUBBLESORT will start with the grey node placed after the white node. As BUBBLESORT (like SWAPS) only ever exchanges adjacent nodes, if it starts from such a scenario, it will never reach a configuration with the white node before the grey, and thus will only reach a costly local optimum. This can also be explained by the fact that BUBBLESORT is simply a method to choose SWAPS-type steps.

The EADES algorithms

Figure 4.13 shows a modification to the standard bad example of Section 4.2.3, in which the optimum solution has two back-arcs: from nodes $n-1$ and n to node 1. Displayed below each node is its indegree. The EADES and EADES IMPROVED algorithms both place node 2 at the left of their solution (as it has the



(a) Global Optimum

(b) Local Optimum

Figure 4.14: Counterexample family (n even) for the Moves local-search heuristic (and thus for Chanas & Chanas Both). Consists of two sets of $n/2$ vertices (black and white), each of which is internally transitive. (a) Shows the global optimum. The black vertices are placed before the white, and each white vertex has a back-arc to the single black vertex $n/2$ positions preceding it. (b) If we interleave the black and white vertices (each set is internally in the same order), we have a locally optimal solution. Yet now there is a back-arc from each black vertex to the white vertex that is $3, 5, 7, \ldots$ positions preceding it.

lowest indegree); with that node removed, the induced subgraph is precisely the same as the original one, albeit one node smaller. The final order will therefore be $(2, 3, 4, 5, \ldots, n-1, n, 1)$, which has a cost of $n - 3$.

Moves and Chanas

The configuration of Figure 4.14(b) is a local optimum for the Moves heuristic. The spacing of the back-arcs ensures that it is never an improvement to move a single node.

Following the discussion at the end of Section 4.2.1, neither the Chanas nor the Chanas Both algorithms could escape from this configuration. Note that this is a local plateau—every possible move will increase the cost. Thus, even repeated calls to Chanas will not remove us from this configuration.

The cost of the local optimum is $n^2/8 - n/4$. The global minimum, shown in Figure 4.14(a), places all black nodes before all white nodes, without changing the relative order within the colour group, thus incurring a cost of $n/2$. So the *locality gap* here is in $\Omega(n)$.

4.3 Min 2-Correlation Clustering

4.3.1 The Algorithms

Local Search for Min-2-CC.

For the Min-2-CC problem, the obvious local improvement to make to is to move (or as we call it, *toss*) a vertex from one cluster to the other if, by doing so, the cost of the clustering is lowered.

Given a clustering $\mathcal{C}$, the clustering $\mathcal{C}_v$ represents the same clustering as $\mathcal{C}$, except with $v \in V$ in the opposite cluster. Remembering that for two clusters, we label vertices with ± 1, we have

$$\mathcal{C}_v(u) = \begin{cases} -\mathcal{C}(u) & \text{if } u = v, \\ \mathcal{C}(u) & \text{otherwise.} \end{cases}$$

We then define $\lambda_v = \text{Min-2-CC}(\mathcal{C}) - \text{Min-2-CC}(\mathcal{C}_v)$, the improvement caused by the change (which is positive if there is some improvement). We define Tosses as follows:

Algorithm: Tosses

Run Local Search with the following neighbourhood relation:

$$\mathcal{N}(\mathcal{C}) = \{\mathcal{C}_v : v \in V\}$$

For any clustering $\mathcal{C}$, we write as $\tilde{\mathcal{C}}$ the locally-optimal clustering that is obtain by running Tosses starting at $\mathcal{C}$. The Tosses algorithm will take time $O(n^3)$ in the worst case.

Pick-a-Vertex Type Algorithms

The PAST Algorithm In Chapter 2, we discussed the Pick-a-Vertex algorithm. It was designed for complete graphs, and there is no obvious extension to incomplete graphs. There may not be a candidate vertex v that is adjacent to every other vertex.

In the complete case, the edges incident to v form a *spanning tree* of the underlying graph G. Now, any spanning tree T induces a clustering of V:

Definition 14. *Let T be a spanning tree of a signed graph $G = (V, E, l)$. Then the clustering $\mathcal{C}^T$ is the unique 2-clustering that agrees with T.*

We can find $\mathcal{C}^T$ by arbitrarily rooting the tree at some vertex $r \in V$, setting $\mathcal{C}^T(r) = 1$ and then deciding $\mathcal{C}^T(v)$ based on the number of $-$ edges on the unique path from r to v. As T is a tree, there are no cycles and every edge is respected. As T spans G, $\mathcal{C}^T$ is properly defined[1].

From this perspective, the PICK-A-VERTEX algorithm is selecting from a set of spanning tree-based clusterings. The spanning trees the PICK-A-VERTEX considers are 'star' trees rooted at each vertex. We therefore propose the PAST (Pick-a-Spanning-Tree) algorithm:

Algorithm: PAST

> For each vertex $v \in V$, perform a breadth-first search from v to find a spanning tree T_v, and the clustering $\mathcal{C}^{T_v}$. Return the $\mathcal{C}^{T_v}$ that minimises MIN-2-CC($\mathcal{C}^{T_v}$).

By using breadth-first-search trees, PAST chooses the same spanning trees as PICK-A-VERTEX on complete graphs, and is thus a generalisation. This time the running time will be $O(n^3)$, as we have an extra step of calculating a BFS tree.

The PASTA-TOSS Algorithm It is certainly possible for the solution chosen by the PAST algorithm to be very much sub-optimal; such a solution can often by improved by a LOCAL SEARCH algorithm such as TOSSES. In fact, we will see such an example in Section 4.3.3. In that section we will also see configurations in which the TOSSES algorithm alone can get stuck at far from optimal configurations.

With this in mind, we could perhaps combine the two algorithms fruitfully. We define PASTA-TOSS, a combination of the two:

[1] Obviously, if we had chosen $\mathcal{C}^T(r) = -1$, we would have obtained a different clustering *function*. However, we would still have reached the same *clustering*. See the end of Section 2.2 for a discussion of this issue.

Algorithm: PASTA-TOSS

> For each vertex $v \in V$, generate the clustering $\mathcal{C}^{T_v}$ as in the algorithm PAST and run TOSSES on $\mathcal{C}^{T_v}$ to get a locally optimal clustering $\widetilde{\mathcal{C}^{T_v}}$. Return the $\widetilde{\mathcal{C}^{T_v}}$ that minimises MIN-2-CC($\widetilde{\mathcal{C}^{T_v}}$).

The combination of algorithms leads to a $O(n^4)$ algorithm, although we'd expect much better performance in practice a one step is a local search. Clearly the PASTA-TOSS algorithm will return a solution no worse than the PAST algorithm. On complete graphs, we know that PICK-A-VERTEX is a 3-approximation. In Section 4.3.3 we will show that PASTA-TOSS is a 2-approximation. Thus the addition of a local-search step to the PICK-A-VERTEX algorithm has substantially improved its approximation factor. This is a significant result, as we have seen that it is rare for local-search algorithms to have any type of approximation guarantees. In fact, we will see that the TOSSES algorithm alone is not a constant-factor approximation algorithm for *any* constant.

The PASTA-FLIP Algorithm

The PASTA-FLIP algorithm is similar to the TRIANGLE DELETION algorithms, but is designed for the MIN-2-CC problem, and can work on incomplete signed graphs—it requires some special techniques to do so.

Removing Bad Cycles We will be operating in a similar way, by reversing the labelling on edges in order to try to remove all 'bad' cycles from the graph. However, in the MIN-2-CC case, it is not $\vec{\triangle}$s that we need to delete, it is a certain type of labelled cycle.

Definition 15. *A* bad cycle *is a cycle C in G in which there is an odd number of negative-labelled edges.*

These cycles are called *bad* as there is no clustering of the vertices in C that satisfies all the labels of C's edges. Like directed cycles, if there are no bad cycles in a graph, solving the problem is easy:

Lemma 2. *Suppose l is a labelling that causes G to have no bad cycles. Then there is a clustering of the vertices with cost zero.*

Proof. Choose some vertex v and assign every other vertex u to a cluster based on the parity of the number of negative edges on the paths between u and v. Note that the parity is uniquely defined: if not, there would be a cycle with an odd number of negative edges. This proves the lemma, as all paths (and edges, as length one paths) are respected. □

The basic principle of PASTA-FLIP is similar to that of the TRIANGLE DELETION algorithms—it might be a good idea to flip the label on an edge that is involved in many bad cycles: those cycles would then become good. If this process could be repeated in an organised way so that no bad cycles remained, then the clustering problem would be trivial.

In the MIN-FAS case, as we were operating on tournaments, we knew that every directed cycle must contain a directed triangle ($\vec{\triangle}$)—a directed cycle of length three. This fact led us to focus on only $\vec{\triangle}$s—if we eliminate all $\vec{\triangle}$s, we eliminate all directed cycles—and there are only $O(n^3)$ such cycles.

However, as we aim to design an algorithm that can operate on incomplete signed graphs, we do not have the luxury of focusing on length-three bad cycles in the MIN-2-CC case. This is a major drawback, as we cannot consider all possible cycles in the graph (there is an exponential number of them). Also, it is conceivable that there is a scenario where there exists a bad cycle, yet there is no edge to flip that will reduce the number of bad cycles. To deal with this problem, we will consider subsets of all the cycles which generate the complete cycle set—much as the $\vec{\triangle}$s generated all the directed cycles. The smallest cardinality such sets are known as cycle bases.

Cycle Bases Let us represent a set of edges by an $|E|$-dimensional vector with entries in $\mathbb{Z}_2$. The sets of edges $\mathcal{E}$ then form a vector space, under the symmetric difference operation. The set of all cycles $\mathcal{F}$ is a subspace of this vector space. Standard results (see e.g. Diestel and Diestel [35] pp. 26-27) show that the cycle space has dimension $|E| - |V| + 1$—that is, it can be generated by certain sets of $|E| - |V| + 1$ independent cycles—such a set is thus a *cycle basis*.

Remark 4. *Let B be a set of cycles of a signed graph G, such that B spans the cycle space $\mathcal{F}$ inside $\mathcal{E}$. Then if no cycle in B is bad, then no cycle in $\mathcal{F}$ is bad.*

Proof. Suppose $C \in \mathcal{F}$ is a cycle. Then C is generated by a set $(b_1, \ldots, b_\ell)$ of cycles from B, none of which are bad. So each cycle b_i has an even number of $-$ edges. As any edge in C is contained in an odd number of b_i, a simple parity argument shows that C contains an even number of $-$ edges. □

A cycle basis spans the complete cycle space, so we can see that if all cycles in a basis are good, Remark 4 tells us we can solve the MIN-2-CC problem trivially. Note that the set of $\vec{\triangle}$s was *not* a basis for the set of directed cycles of a tournament, it did span them (as evidenced by Figure 2.6).

Although there are many cycle bases, we can generate one for any spanning tree T of G. We obtain a basis known as a *fundamental basis* with respect to T by forming a cycle C_e for each edge $e = (v, w) \notin T$: C_e is e plus the path in T from v to w.

The PASTA-FLIP Algorithm Consider such a fundamental cycle basis B^T, for some spanning tree T. Each edge $e \notin T$ will only be involved in a single cycle in the basis, C_e. So there is one straightforward technique to ensure that each cycle in the basis is good: if C_e is bad, simply flip e. At the end of this process, the labellings on the T-edges will be respected. So if we had started with T^v for some vertex v, we have described precisely the per-vertex operation of the PAST algorithm.

However this is inefficient—edges inside T are involved in many basis cycles, whereas each other edge is only involved in one. Flipping one of these edges could potentially fix a number of bad cycles (in the basis), and thus mean fewer flips. This is similar to the idea of TRIANGLE DELETION algorithms. Each flip represents a disagreement between the output clustering and the (original) edge labelling. With this in mind, we define the PASTA-FLIP (PICK-A-SPANNING TREE AND FLIP) algorithm as follows:

Algorithm: PASTA-FLIP

> For each vertex $v \in V$ create a breadth-first-search tree T_v, and the corresponding fundamental basis B^{T_v}. While there is an edge $e \in T_v$ which is involved in more bad basis cycles than good, flip e. When there are no such edges left, produce the clustering induced by the (probably altered) T_v, $\mathcal{C}^{T_v}$. Return the solution found of lowest MIN-2-CC cost.

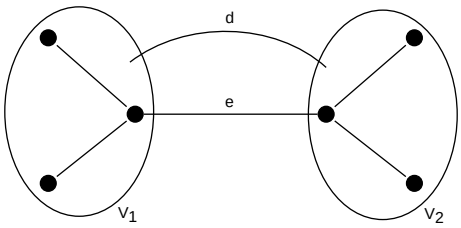


Figure 4.15: Flipping an edge in PASTA-FLIP.

The action of "flipping" never worsens the MIN-2-CC cost, leading to a worst-case runnning time of $O(n^4)$, and giving the following theorem.

Theorem 1. *The cost of the solution returned by* PASTA-FLIP *is no greater than the cost of the solution returned by* PAST.

Proof. Consider some particular spanning tree T. We will show that PASTA-FLIP will not produce a worse clustering than PAST would on T. Considering the operation of both algorithms, this will be sufficient to prove the theorem.

The MIN-2-CC solution induced by PAST is completely determined by the sign of the edges in the spanning tree. The only difference for PASTA-FLIP is when an edge is flipped. However, we claim each edge flip is a *local semi-improvement* step—that is, the cost of the labelling induced after the flip is always less than or equal to the cost before. Thus PASTA-FLIP, by induction, will have lower or equal cost to PAST.

To see that an edge flip is a local semi-improvement, consider Figure 4.15. Here we are considering flipping an edge e on the spanning tree T. As T is a tree, e separates V into two subsets V_1 and V_2. Flipping e will flip all the labels inside one subset, and thus invert the 'correctness' of each edge $d \in D = E_c(V_1, V_2) \setminus e$, the set of edges linking vertices between V_1 and V_2. So, if $\mathcal{C}$ is the current clustering induced by the spanning tree, and $\mathcal{C}'$ is the solution induced after e is flipped,

$$\text{MIN-2-CC}(\mathcal{C}') \leq \text{MIN-2-CC}(\mathcal{C}) + |D_{\text{correct}}| - |D_{\text{incorrect}}| + 1$$

Here $D_{\text{correct}} = \{d \in D, d \text{ satisfied by } \mathcal{C}\}$ are the edges between V_1 and V_2 which

will be made incorrect by flipping e, and $D_{\text{incorrect}}$ is defined in the analogous way. The final term (the $+1$) and the inequality is due to the fact that reversing e might violate e itself.

So we need to show that e will only get flipped if $|D_{\text{correct}}| \leq |D_{\text{incorrect}}|$. For any $d \in D$, as $d \notin T$, there is a cycle $C_d \in B^T$. Clearly C_d contains e, as we know e separates V_1 and V_2 on T. Also it is clear that any cycle through e in the basis must contain exactly one such D-edge.

Suppose that d is not satisfied by $\mathcal{C}$. This means that the total sign of the path $C_d \setminus d$ is different to that of d—that is, C_d has an odd number of $-$ edges. So C_d is a bad cycle. Conversely, if d is satisfied, C_d is good. So the number of bad cycles that e is involved in is exactly the number of unsatisfied D edges ($|D_{\text{incorrect}}|$). This completes the proof as we know the algorithm will only flip e if that number is strictly greater than the number of good cycles that e is involved in ($|D_{\text{correct}}|$). □

4.3.2 Experiments

In this section, we perform some experimental tests of the MIN-2-CC algorithms we have described in this chapter, along with some mentioned previously. We aim to obtain an idea of the practical performance of the algorithms, to complement the theoretical results of the next section.

Algorithms Tested

In this section, we will test all of the algorithms described in Section 4.3.1, along with the following additional algorithms:

- The algorithm of Dasgupta et al. [28], based on a Goemans-Williamson style SDP, which we term GW-SDP.
- The spectral MIN-2-CC relaxation, as discussed in on Page 87 of Chapter 3.

As mentioned, the PTAS of Guruswami and Giotis [53] was not feasible to implement, so we tested a PTAS-like algorithm, called PTAS-m, where m is the sample size. Also, the algorithm we refer to below as TOSSES takes $n = |V|$ randomised starting clusterings, and produces the best clustering found after toss-based search

from each. This is to compare it to PASTA-TOSS, which uses n PAST-style starting clusterings. Also we experimented with PASTA-FLIP+TOSS, which performs PAST, and for each tree flips edges (until no more flips are possible) and then tosses vertices.

Datasets

For our experimental work, we used three datasets: the regulatory network of human epidermal growth factor (**EGFR**), as used by both Dasgupta et al. [28] and Huffner et al. [59] in their investigations, and two synthetic datasets.

Each synthetic dataset was generated randomly subject to two parameters, which were independent over each edge. The first, p_e, is the probability that an edge exists (with either sign), and given the edge exists. The second, p, is the probability that the edge agrees with a randomly-generated initial clustering.

The first data set, called **SPARSE**, had problems of size 200, a p_e value of 0.05, and a p value of 0.3, which was an attempt to approximate the **EGFR** dataset. The second data set, called **COMPLETE**, had problems of size 100, $p_e = 1$—thus all graphs are complete—and $p = 0.45$. We found, empirically, that lower values of p result in MIN-2-CC problems that are easier to solve.

All experiments were run on a 2 GHz Intel Core 2 Duo machine with 2GB of RAM, running MAC OS. All algorithms were implemented in C, apart from SPECTRAL CLUSTERING and GW-SDP, which were run in Matlab 7.4.0. Note that this means the times recorded for the Matlab algorithms perhaps were not entirely appropriate for comparison.

Results

Figures 4.16 and 4.17 show the relative performances of the algorithms discussed in this paper on the **EGFR** and **COMPLETE** datasets. These plots compare the algorithmic performance (average number of errors; that is the CORRELATION CLUSTERING cost) to the average time taken to achieve that performance. As we can see, the PTAS algorithms and PAST perform poorly as a rule; DASGUPTA can achieve good results, but is very slow in comparison to our algorithms. Although the SPECTRAL CLUSTERING technique can be quite fast on sparse graphs, its performance is

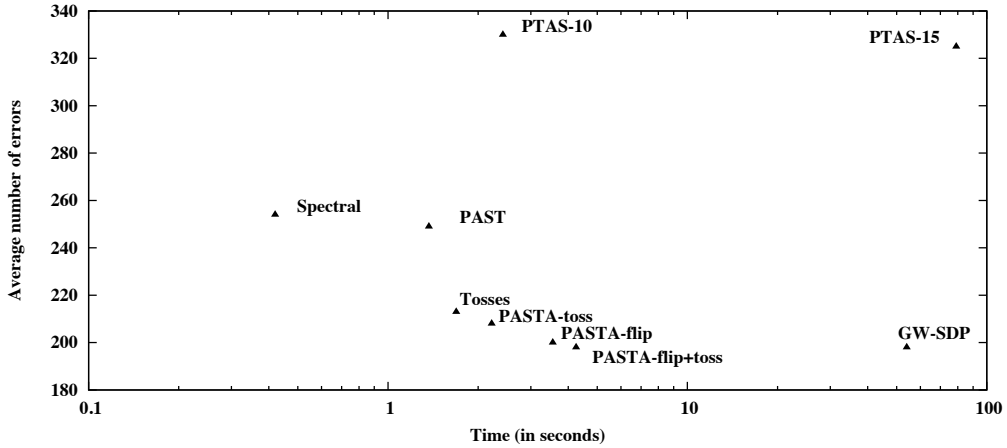


Figure 4.16: The time/effectiveness profile on the **EGFR** Dataset.

not great.

The more interesting result is the good performance of the TOSSES heuristic: even though in the next section we will see that the TOSSES algorithm has no theoretical guarantees (there are examples where it performs arbitrarily badly) on the real examples that we consider, it does almost as well as PASTA-TOSS, which does have a strong theoretical guarantee on its performance. This quality of strong performance without theoretical guarantees is common to local-search algorithms; this is a reason why the fact that PASTA-TOSS does have a guarantee is particularly important. Table 4.4 summarises the results on all three datasets.

4.3.3 Theoretical Results

TOSSES

The TOSSES algorithm, used naively, has no constant-factor approximation guarantee.

Consider a complete graph with $n/2$ disjoint edges labelled $-$, with all other edges are labelled $+$, as shown in Figure 4.18. The global minimum here has cost $n/2$; however, there is a local minimum—which cuts across each minus-edge—that has cost $n(n-2)/4$.

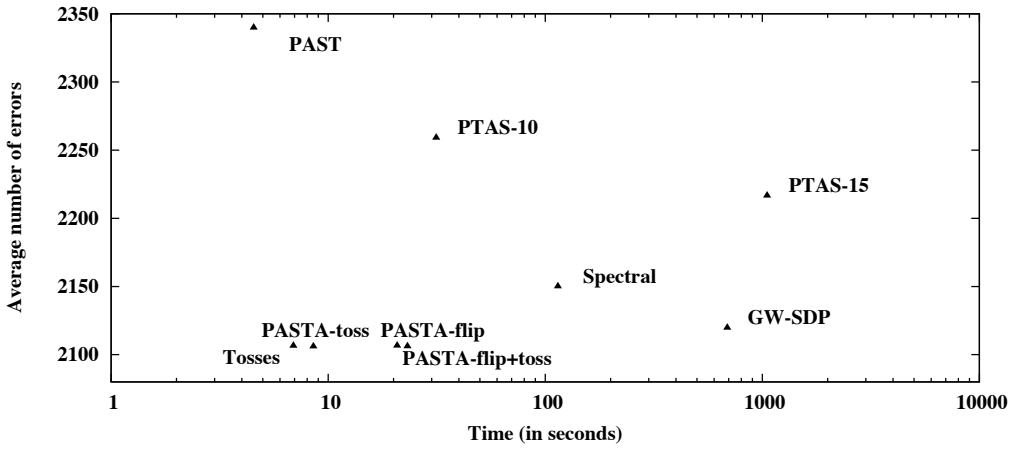


Figure 4.17: The time/effectiveness profile on an average over 100 instances of *complete* signed graphs, $n = 100$ and $p = 0.45$.

	EGFR		Sparse		Complete	
Algorithm	Cost	Time	Cost	Time	Cost	Time
GW-SDP	-7.042	54.01	-1.081	26.28	0.626	689.01
Spectral Clustering	19.249	0.42	22.413	2847.70	2.064	114.35
PAST	16.901	1.37	29.756	29.16	11.071	4.54
PASTA-flip	-6.103	3.55	-3.338	80.60	0.004	20.77
PASTA-toss	-2.347	2.22	-1.990	54.82	-0.026	8.54
PASTA-flip+toss	-7.042	4.25	-4.184	99.62	-0.022	23.18
PTAS-10	54.930	2.42	49.897	102.97	7.239	31.45
PTAS-15	52.582	79.06	40.525	3407.03	5.228	1052.27
Tosses	0.000	1.69	0.000	48.03	0.000	6.91

Table 4.4: The results of running all Min-2-CC algorithms on all datasets. We report the average of the percentage (%) relative difference between the number of errors compared to Tosses, over all problem instances in the dataset. The running time is measured in seconds.

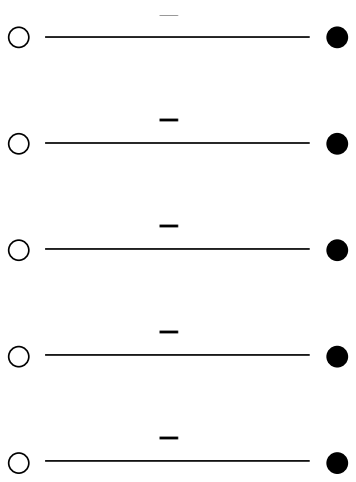


Figure 4.18: A counter-example for the TOSSES algorithm. This is a complete signed graph—all edges not shown are labelled +. The local minimum places the black and white vertices in different clusters, with cost 20, whilst the global minimum places all vertices together, with cost 5.

PICK-A-VERTEX

Bansal et al. proved that PICK-A-VERTEX was a 3-approximation for the MIN-2-CC problem on complete graphs. It turns out that the 3-approximation is tight, a fact not mentioned in the original paper. Consider a *complete* graph, consisting solely of positive edges, apart from a Hamiltonian cycle of negative edges, as shown in Figure 4.19. The optimal solution (placing all vertices together) has cost n, whilst every PICK-A-VERTEX solution will have cost $3n - 10$. Notice that the PICK-A-VERTEX clustering described above is *not* a local optimum.

4.3.4 Proof that PASTA-TOSS is a 2-approximation

In this section we develop theoretical results, leading to a proof that PASTA-TOSS is a 2-approximation on complete graphs. This is the main theoretical result of this chapter; as we discussed earlier, theoretical results about local-search algorithms are rare. We have already seen that TOSSES alone has no constant-factor guarantees. To begin, we need the concept of a switching class.

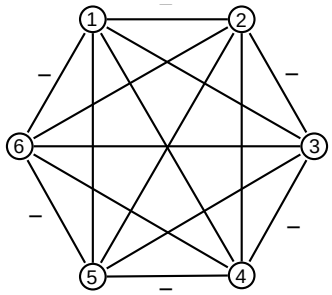


Figure 4.19: A counter-example for the Pick-a-Vertex algorithm. All unlabelled edges are labelled +. The optimal solution places all nodes together, with cost 6. As the graph is symmetric, it does not matter which vertex is 'picked'. Suppose 1 is chosen—the resulting clustering is $(\{1, 3, 4, 5\}, \{2, 6\})$, which has cost 8.

Switching

The notion of *switching* in signed graphs is well established [111]. Given a labelling l, we generate another labelling l_v by selecting a vertex v and flipping the labels on the edges incident to v. We may repeat this switching operation at other vertices, generating further labellings. The family of all possible labellings can be partitioned into equivalence classes under this (multiple) switching operation: we refer to labellings in the same class as *switching equivalent*.

We also introduce the notion of switching on 2-clusterings: we *switch* a clustering $\mathcal{C}$ to $\mathcal{C}_v$ by tossing v to the other cluster. We have seen this earlier in the definition of the Tosses algorithm. In this way, every clustering can be obtained by a series of switching steps from $\mathcal{C}$. Note that this same fact *does not* hold about labellings.

Lemma 3. *The cost of $\mathcal{C}$ under l is the same as the cost of $\mathcal{C}_v$ under l_v.*

Proof. The only edges affected by these operations are edges incident to v (see Figure 4.20). For such an edge $e = (v, u)$, $l(e) = -l_v(e)$, and $\mathcal{C}(v) = -\mathcal{C}_v(v)$, whilst $\mathcal{C}(u) = \mathcal{C}_v(u)$, so the cost of this edge is unchanged. □

Lemma 3 tells us that if l has a solution of cost k, l_v also has a solution of cost k. Also as $(l_v)_v = l$, the converse is true. Consequently, we see the following useful corollaries.

Corollary 3. *Let $G = (V, E, l)$ be a signed graph. Then, for all labellings l' in l's switching class, the optimal* Min-2-CC *cost on $G' = (V, E, l')$ is equal to the the*

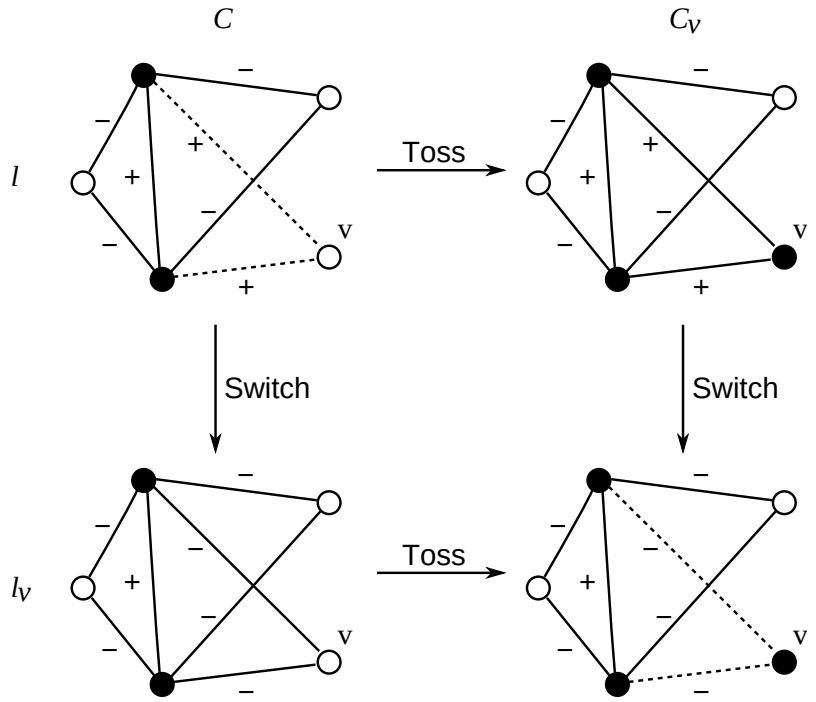


Figure 4.20: A demonstration that switching and tossing at the same vertex does not affect the edges that are violated. The dashed edges are not respected by the relevant clustering ($\mathcal{C}$ on the left and $\mathcal{C}_v$ on the right). We see that the same edges are violated by $\mathcal{C}_v$ under l_v as by $\mathcal{C}$ under l.

optimal Min-2-CC *cost on G. In particular, if $\mathcal{C}^*$ is an optimal clustering for l, then $\mathcal{C}^*_v$ is an optimal clustering for l_v.*

Corollary 4. *For a given labelling l of a graph $G = (V, E, l)$, there exists a labelling $\bar{l}$, switching equivalent to l, for which placing all vertices together in one cluster is optimal.*

Proof. Suppose $\mathcal{C}$ is the optimal solution to Min-2-CC on G. Let $S = (v_1, \ldots, v_\ell)$ be the set of vertices labelled -1 by $\mathcal{C}$. If we let $\mathcal{C}^0 = \mathcal{C}$, and $\mathcal{C}^i = (\mathcal{C}^{i-1})_{v_i}$, then $\mathcal{C}^\ell$ places all points together in one cluster. Consider the switching equivalent labellings l^i defined in a similar fashion. Then Corollary 3 tells us that $\mathcal{C}^i$ is the optimal clustering for $(V, E, l^i = \bar{l})$. □

Note that the optimal cost for $\bar{l}$ in Corollary 4 equals the number of negative edges in $\bar{l}$.

Switching-Invariant Algorithms

Imagine we knew that an algorithm behaved in essentially the same way on all switching-equivalent labellings. Then Corollary 4 tells us that we can focus on labellings in which the optimum has all elements in one cluster.

Definition 16. *An algorithm is* switching invariant *if, whenever it produces $\mathcal{C}$ on input l, it produces $\mathcal{C}_v$ on input l_v.*

We now investigate the behaviour of two key algorithms under switching.

Lemma 4. PAST *is switching invariant.*

Proof. Consider a switch on $v \in V$. Let T be any spanning tree of G inducing a clustering $\mathcal{C}^T$ under PAST. Consider two vertices u and x in V. Whether they are clustered together depends only on the parity of the number of negative edges on the path between u and x in tree T. If v is not on this path, clearly the parity is unchanged. If v is on the path, the parity is changed only if u or x is v.

So, under l_v, the clustering based on T is switching invariant. Lemma 3 tells us that the spanning tree that induces the best clustering on l also induces the best clustering on l_v. □

We can now infer an interesting fact about spanning trees.

Lemma 5. *For a given labelling l on a graph G, there exists a spanning tree T that induces an optimal clustering.*

Proof. Consider labelling $\bar{l}$ as defined in Corollary 4. The positively-labelled edges in $\bar{l}$ form a subgraph that is connected and spans G: if they did not, then there would be a non-trivial cut of G with only $\bar{l}$-negative edges. This would imply that the optimum clustering must use two clusters, contradicting the definition of $\bar{l}$. Hence, we can find a spanning tree T of positively-labelled edges in $\bar{l}$: this induces a solution with all vertices in one cluster. The proof of Lemma 4, combined with Corollary 3 shows that T will induce the optimum solution on l. □

TOSSES is not exactly switching invariant, but we can prove a similar result.

Lemma 6. *If* TOSSES *is given l as input and uses $\mathcal{C}$ as a starting point, resulting in solution $\widetilde{\mathcal{C}}$, then given input l_v and starting point $\mathcal{C}_v$,* TOSSES *produces solution $(\widetilde{\mathcal{C}})_v$.*

Proof. Consider running two simultaneous instances of TOSSES, one starting from $\mathcal{C}$ and the other starting from $\mathcal{C}_v$. To begin with, the only edges that could possibly be different are the edges incident to v. The proof of Lemma 3 shows that the edges that incur a cost are the same in $(l_v, \mathcal{C}_v)$ as they are in $(l, \mathcal{C})$. So in both cases, λ_u is the same, for all $u \in V$. Therefore the same sequence of vertices will be chosen to be tossed. □

The following lemma is an immediate consequence of Lemmas 4 and 6.

Lemma 7. PASTA-TOSS *is switching invariant.*

Proof that PASTA-TOSS is a 2-approximation

Since PASTA-TOSS is switching invariant, we can analyse its behaviour on input labellings in which the optimum places all vertices in one cluster (refer to Corollaries 3 and 4). For such a labelling, no vertex has minus-degree more than $n/2$, as otherwise a better clustering would be to place that vertex in its own cluster. Also, the optimum cost is simply the total number of minus-edges in the graph,

as we know the optimal solution does not break an edge. If we let β be the minimum of the minus-degree (number of incident $-$ edges) of all the vertices, then MIN-2-CC$^* \geq \beta n/2$.

To analyse the performance of PASTA-TOSS on complete graphs, consider the iteration where PASTA-TOSS uses the spanning tree from v, a node of minus-degree β. Initially, the algorithm splits the vertices into two sets, $X_0 = \{v\} \cup N^+(v)$ and $Y_0 = N^-(v)$. As the local-search algorithm progresses, vertices will be tossed from one set to the other (call them X and Y). Consider the point at which the first vertex is tossed from X to Y. Note that this means $|Y| \leq \beta$.

We can compare the cost of the clustering (X, Y) to MIN-2-CC* in a fashion similar to Bansal et al. We form an estimate of the difference in cost between our clustering and an optimum by counting the number of $+$ edges between X and Y, discounting the $-$ edges. For a vertex $v \in V$, and a set A, define A_v^+ to be the number of $+$ edges from v to A. A_v^- is defined in the analogous way. Then

$$\text{MIN-2-CC}(X,Y) - \text{MIN-2-CC}^* = \sum_y X_y^+ - \sum_y X_y^- = \sum_y \text{pull}_y \leq \beta \max_{y \in Y} \text{pull}_y \,, \tag{4.1}$$

where $\text{pull}_y = X_y^+ - X_y^-$ is the "pull" that X exerts on y.

If we let $\text{push}_y = Y_y^- - Y_y^+$ (the "push" that Y exerts on y), the local improvement of swapping any node $y \in Y$ is given by

$$\text{imp}_y = \text{pull}_y + \text{push}_y \tag{4.2}$$

So we can use a bound on the local improvement of swapping a node (from X) to get a contradictory bound on pull_y for a given y.

Theorem 2. PASTA-TOSS *is a* 2*-approximation on complete graphs.*

Proof. We claim at this point, when the first node to be swapped is from X, that MIN-2-CC$(X, Y) \leq 2$(MIN-2-CC*).

Arguing by contradiction, suppose that MIN-2-CC(X, Y) is not within a factor 2 of MIN-2-CC*. Then there must be some $y_0 \in Y$ such that $\text{pull}_{y_0} > n/2$.

Let x_0 be the vertex from X that is about to be swapped. By definition,

$$Y^+_{x_0} + Y^-_{x_0} = |Y| \text{ and } X^+_{x_0} + X^-_{x_0} = n - |Y| - 1$$

Since we have assumed that the optimum solution places all vertices together, x_0 is incident to at most $n/2$ negative edges, and so $X^-_{x_0} \leq n/2$. So we have

$$\text{imp}_{x_0} = X^-_{x_0} + Y^+_{x_0} - X^+_{x_0} - Y^-_{x_0} \leq 2X^-_{x_0} - (X^-_{x_0} + X^+_{x_0}) + (Y^+_{x_0} + Y^-_{x_0}),$$

which is at most $2|Y| + 1$. Given x_0 is the vertex which is about to be swapped, $\text{imp}_{y_0} \leq \text{imp}_{x_0} \leq 2|Y| + 1$.

Alternatively, if the algorithm ends without ever swapping an $x \in X$, at the conclusion, $\text{imp}_{y_0} \leq 0 < 2|Y| + 1$.

So, using (4.2) and our assumption, we have

$$\text{push}_{y_0} = \text{imp}_{y_0} - \text{pull}_{y_0} < 2|Y| - 1 - n/2 \tag{4.3}$$

Now we use the fact that y_0 has to have at least β minus-edges to show a contradictory lower-bound on push_{y_0}. We have

$$\begin{aligned}
\text{push}_{y_0} &= Y^-_{y_0} - Y^+_{y_0} \\
&= 2Y^-_{y_0} - |Y| + 1 \\
&\geq |Y| - 2X^-_{y_0} + 1 \\
&= |Y| + X^+_{y_0} - X^-_{y_0} - (X^+_{y_0} + X^-_{y_0}) + 1 \\
&> 2|Y| + 1 - n/2
\end{aligned}$$

The first equality follows as $Y^-_{y_0} + Y^+_{y_0} = |Y| - 1$, the first inequality as the minus-degree of y_0, $Y^-_{y_0} + X^-_{y_0}$ is at least $\beta \geq |Y|$, and the second as $X^+_{y_0} + X^-_{y_0} = n - |Y|$ and $X^+_{y_0} - X^-_{y_0} = \text{pull}_{y_0} > n/2$. □

4.4 Conclusion

In this chapter, we have outlined a number of algorithms for both the Min Feedback Arc Set and the Min 2-Correlation Clustering problems. We studied those algorithms, along with the state of the art algorithms for these problems, with reference to both practical issues, and theoretical guarantees.

Notably, we have found that, for the Min-FAS problem, the Chanas algorithm performs very well in practice. Additionally it has the advantage of the ability to be combined with other algorithms; the best such algorithm to combine it with seems to be the Degree Difference Sampled 1 algorithm, defined here. However, as it is a variant of the Moves local-search heuristic, we have seen that it unfortunately has no constant-factor approximation guarantees.

For the Min-2-CC problem, the algorithm PASTA-toss, as developed in this chapter, achieves the strongest experimental results, along with the similar PASTA-flip algorithm. Additionally, we have seen that PASTA-toss is a 2-approximation algorithm, the best known constant-factor approximation result for the Min-2-CC problem. A PTAS exists, but we have seen that in practice it is not very effective. This approximation guarantee, rare for local-search approaches, is another point in favour of this algorithm.

Chapter 5

Sampling

The final technique for solving approximation problems we are concerned with is *sampling*. The idea of sampling is to take a large, complicated instance of a problem, and find a less complex sub-problem which represents the larger one, by choosing—sampling—a small set of vertices. We can then quickly solve the small, sampled instance, quickly and often exactly, and somehow use the smaller solution to build up a larger, complete solution to the original instance.

If we can somehow relate the size of the sampled sub-problem to the quality of the solution to the large problem, then we can use this technique to trade-off running-time with solution quality. If we take a larger sample, it will take a longer time to find a very good solution to the sampled solution, but we will know we get a better large-scale solution. It is this trade-off which is exploited by many polynomial-time approximation schemes (PTASes).

As we've seen before, sampling algorithms can sometimes be specified with sample sizes so unrealistic that they become infeasible. However, this does not mean that sampling cannot be used effectively for practical algorithms. For instance, Bertolacci and Wirth [17] find that using a sampling technique, they can increase the size of Correlation Clustering problems they can tackle using a variety of algorithms.

In this chapter we will present PTASes for the k-Consensus Clustering problems. These algorithms will exploit a PTAS designed for the Max-k-Cut problem, which uses such sampling heavily. Additionally, it exploits the related idea of *repre-*

sentative sampling, which involves sampling points which 'capture' the location of a cluster in some way. This related the superset sampling idea of Kumar et al. [74] for *geometric* problems.

5.1 Introduction

In this chapter, we will present a series of results that will lead us to a Polynomial Time Approximation Scheme for the k-Consensus Clustering problem. Note that this is the standard Consensus Clustering problem, with the added restriction that the output clustering can have no more than k clusters, where k is a constant.

The reason that we pursue the k-Consensus Clustering problem, and not the full-blown Consensus Clustering, in seeking a PTAS is that Bonizzoni et al. [18] recently demonstrated that min-Consensus Clustering is APX-hard, even with as few as 3 input clusterings. This means that, unless P = NP, no PTAS for min-Consensus Clustering with unrestricted number of clusters exists. Considering the success of Giotis and Guruswami [53] in finding a PTAS for the 0/1 Min-k-CC problem—again, in the 0/1 case, the Min-CC problem is APX-hard [21]—it is natural to consider the restriction to min-k-Consensus Clustering.

Connecting Consensus Clustering to Max-Cut. As we outlined in Section 2.2.6, any instance of the Consensus Clustering problem can be transformed into an instance of metric-Correlation Clustering. So, rather than designing a PTAS for min-k-Consensus Clustering, in this chapter we will focus on constructing a PTAS for the more general metric-Min-k-CC problem. In order to do that, we will exploit the relationship between k-CC and the k-Cut problems; that is Min-k-Uncut and Max-k-Cut. This relationship, as we mentioned in Section 2.2.4, is

$$\text{Max-}k\text{-CC}(\mathcal{C}) = \text{Max-}k\text{-Cut}(\mathcal{C}) + e_u(\mathcal{C}) - \text{Min-}k\text{-Uncut}(\mathcal{C}) \tag{5.1a}$$

$$\text{Min-}k\text{-CC}(\mathcal{C}) = \text{Min-}k\text{-Uncut}(\mathcal{C}) + e_c(\mathcal{C}) - \text{Max-}k\text{-Cut}(\mathcal{C})\,, \tag{5.1b}$$

The reason we are interested in this relationship is because, as we mentioned in Section 2.2.2, the k-Cut problem is very well studied. In fact, there is a PTAS for both metric-Max-k-Cut [44] and metric-Min-k-Uncut [60; 43]. So, in this chapter, we will aim to use those PTASes to construct another for metric-Min-k-CC.

A PTAS for metric Min-k-Uncut. To achieve this, we will need to study the PTAS for metric-Min-k-Uncut in some detail. The PTAS for the general case $k \geq 2$ was developed by Fernandez de la Vega et al. [43], expanding a PTAS originally developed for metric-Min-Uncut, by Indyk [60].

We will describe both PTASes in detail in subsequent sections, but the basic structure of the algorithms is a familiar one for designers of minimisation PTASes. We harness an existing maximisation PTAS for the complementary problem; although this will not work in every instance, as a good maximisation approximation is not necessarily good for the minimisation objective, such instances have special properties that we can exploit. In this case, the maximisation PTAS, for the metric-Max-k-Cut problem was supplied by Fernandez de la Vega and Kenyon [44], building on various dense-Max-k-Cut PTASes, as were mentioned in Section 2.2.2.

Results of this Chapter. In this chapter, we will mirror the results mentioned for the metric k-Cut problems to the metric k-CC problems. That is, we will find a PTAS for the metric-Max-k-CC problem, and then exploit it to develop a PTAS for the metric-Min-k-CC problem. Consequently, we will have a PTAS for both k-Consensus Clustering problems.

Indyk, and subsequently Fernandez de la Vega et al., exploit a 'well-separatedness' property of metric-Min-k-Uncut problem instances that are not approximable by the metric-Max-k-Cut PTAS to show that an algorithm based on representative sampling will be fruitful in the metric-Min-k-Uncut case. In Section 5.4.3, we will show that the metric-Min-k-CC instances that our metric-Max-k-CC PTAS does not approximate well also share *the same* well-separatedness quality, and thus can be approximated from a Min-k-Uncut perspective, using the same algorithm.

We will then use the relationship between k-CC and k-Cut—see (5.1)—which tells us that if we have a good Min-k-Uncut approximation to an optimal Min-k-

CC solution *with the same cluster sizes*, we have a good Min-k-CC approximation. This means that if the algorithm of Fernandez de la Vega et al.—which we can apply as the well-separatedness condition holds—could guarantee generating certain cluster sizes, we could simply apply it to the k-CC case.

However, the algorithm of Fernandez de la Vega et al. makes no such guarantees. So we need to use a very important *assignment* step which will serve to balance the sizes of the clusters correctly. This will ensure than we can apply (5.1), and achieve the minimisation PTAS.

We refer the reader to Section 2.2.4, and to Table 2.1 to place our results in the context of other results regarding Consensus Clustering, Correlation Clustering and Max-Cut.

5.1.1 Preliminaries

As we focus exclusively on metric problems in this chapter, instead of w, we will use δ to indicate the distance between vertices. Then define $\delta(A, B)$ to be $\sum_{\substack{a\in A\\ b\in B}} \delta(a, b)$, and $\delta(A)$ to be $\delta(A, A)$. The following is a simple application of the triangle inequality:

Proposition 3. *For all $X, Y, Z \subseteq V$,*

$$|Y|\delta(X, Z) \leq |Z|\delta(X, Y) + |X|\delta(Y, Z)\,.$$

Proof. For every $x \in X, y \in Y, z \in Z$, we know that

$$\delta(x, z) \leq \delta(x, y) + \delta(y, z)\,.$$

So, summing over X, Y, Z, we have

$$\begin{aligned}|Y|\delta(X, Z) = \sum_{x\in X, y\in Y, z\in Z} \delta(x, z) &\leq \sum_{x\in X, y\in Y, z\in Z} \delta(x, y) + \sum_{x\in X, y\in Y, z\in Z} \delta(z, y)\\ &= |Z|\delta(X, Y) + |X|\delta(Y, Z)\,.\end{aligned}$$

□

5.2 Overview

We begin this chapter by explaining, without proof, how Indyk [60] used the PTAS for Max-Cut, (the maximisation version of k-Cut with $k = 2$) to find a PTAS for Min-Uncut, which is the corresponding minimisation version. We can then sketch an algorithm, an analogue of our full Min-k-CC algorithm, for the case $k = 2$. This will explain many of the techniques involved, and highlight the connections and the extra difficulty involved in the k-CC case.

Using a Max-Cut PTAS for Min-Uncut. As we have seen in Section 2.2.2, there is a long history of PTASes to solve Max-Cut and Max-k-Cut in the *dense* case. Fernandez de la Vega and Kenyon [44] provided a simple, clean reduction to a PTAS for the metric-Max-Cut problem. However, as we discussed in Section 1.1, and will see precisely in Remark 7, we cannot simply apply a maximisation PTAS to a complementary minimisation problem to get a PTAS.

It turns out that we can use the Max-Cut PTAS for Min-Uncut precisely when

$$\text{Max-Cut}^* \leq O(\frac{1}{\epsilon})\text{Min-Uncut}^* \tag{5.2}$$

That is, the PTAS for metric-Max-Cut will only act as a PTAS for those cases of metric-Min-Uncut where the optimal Max-Cut value is not too much bigger than the optimal Min-Uncut value. Or, perhaps of more interest, in the cases where it *does not* work, that Max-Cut value is much larger. Remembering (1.1), we can see that this means the 'difficult' instances are those in which the value Max-Cut* is close to the value $W = \sum_{u,v \in V} \delta(u, v)$.

Indyk's insight was to ask what exactly this implies about those instances of Min-Uncut. If Max-Cut* is very large, then Min-Uncut* is close to zero: the edges cut by the optimal clustering $\mathcal{C}$ (which is of course shared by both problems) tend to be relatively long, and the edges not cut tend to be relatively short. This of course means that the solution is particularly easy to find; in fact a simple sampling algorithm will do the job.

5.2.1 Indyk's Algorithm

Let us assume we are considering an graphs G, where (5.2) does not hold. Consider the optimal clustering, $\mathcal{C}$. To begin with, we will guess the cluster sizes $|C_1|$ and $|C_2|$ (without loss of generality, $|C_1| \geq |C_2|$). We then proceed differently depending on what those sizes are. How can we guess the sizes $|C_1|$ and $|C_2|$ when we have no clue about $\mathcal{C}$? Clearly we cannot, but we can simply run our algorithm for each of the n different possibilities. This will add a multiplicative factor of n to our running time, and it will remain polynomial. In fact, in the full algorithm, we will be guess many such quantities. However, as long as each quantity only adds a polynomial factor, and there are only a constant number of them, we retain a overall polynomial running time.

What we cannot do, for instance, is guess the cluster membership of each point. Although there are only two choices for each point in V, there are n points—adding n different multiplicative factors of 2 leads to an increase in running time of 2^n.

One larger cluster: Unbalanced Cut. Suppose C_1 is much larger than C_2. Then a point c_1 chosen at random is likely to be in C_1. In fact, there is a good chance that c_1 will be a good *representative* of C_1, that is

Definition 17.

$$\delta(\{c_1\}, C_1) \leq \frac{2\delta(C_1)}{|C_1|} \tag{5.3}$$

This means that the distance from a point $v \in V$ to c_1 is a good estimate of the distance from the point to C_1, viz.

Lemma 8 (F. de la Vega et al., Lemma 4)**.**

$$|\delta(v, c_1)|C_1| - \delta(v, C_1)| \leq 2\delta(C_1)/|C_1|\,.$$

As we can now use c_1 as a stand-in for the full cluster C_1, we can simply find the full clustering by the following simple procedure.

Algorithm: Unbalanced Cut

> Let D_1 be the $|C_1|$ closest points to c_1 as one cluster, and D_2 be the remaining points.

Two large clusters: Balanced Cut. Suppose instead that the two clusters are both relatively large. In this case, we should consider the distance to both clusters when cutting—previously we could ignore the distance to C_2 as it was small enough to be irrelevant. To do this, we need a representatives from each clusters—however, as both clusters are large, there is a relatively high probability of any two randomly sampled points being representatives of both clusters. Repeated sampling—basically guessing as mentioned previously—will yield a c_1 and c_2 which are representatives as per Definition 17.

Remembering Lemma 8 above, we define the quantity $\hat{\delta}_i(v) = |C_i|\delta(v, c_i)$. As c_i is a representative of C_i, by Lemma 8, $\hat{\delta}_i(v)$ will approximate the quantity $\delta_i(v) = \delta(\{v\}, C_i)$. Now that we have a good measure of the distance to both clusters, we can run the following algorithm:

Algorithm: Balanced Cut

Output a clustering $\mathcal{D}$, defined as follows:

$$\mathcal{D}(v) = \begin{cases} 1 & \text{if } \hat{\delta}_1(v) \leq \hat{\delta}_2(v); \\ 2 & \text{otherwise.} \end{cases}$$

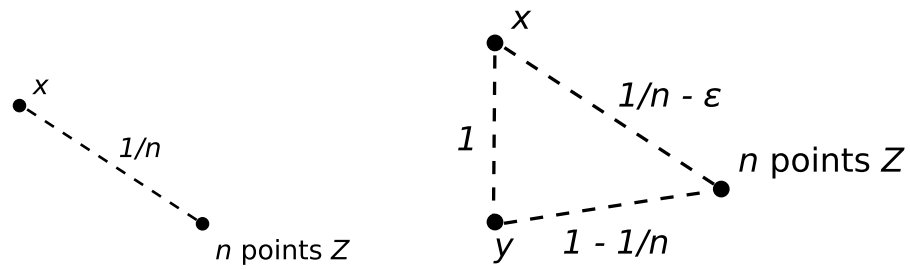
So Indyk's algorithm will use either a) the metric-Max-Cut PTAS, b) Unbalanced Cut, or c) Balanced Cut, depending on whether the clusters in $\mathcal{C}$ are a) close, b) far and different sizes, c) far and both large. Indyk outlines techniques to run his algorithm in time $O(\log^{1/\epsilon^{O(1)}} n^{1+\gamma})$, for any $\gamma > 0$.

5.2.2 Adapting Indyk's Algorithm to Min-CC.

The key observation here is the connection between Min-k-Uncut and Min-k-CC, which using (5.1b) and a generalisation of (1.1) gives, for any clustering $\mathcal{X}$:

$$\text{Min-}k\text{-CC}(\mathcal{X}) = 2\text{Min-}k\text{-Uncut}(\mathcal{X}) + e_c(\mathcal{X}) - W \tag{5.4}$$

As we are dealing with a complete graph (a metric), $e_c(\mathcal{X}) = \sum_{i<j\leq k} |X_i||X_j|$.



(a) A single point x, and n copies of the same point Z. (i.e. $\delta(z_1, z_2) = 0$ for all $z_i \in Z$.)

(b) Single points x and y, and n copies of the same point Z.

Figure 5.1: Examples of the differing behaviour of Min-Uncut and Min-2-CC when cluster sizes are different. For (a), the optimal solutions are different; in (b) the optimae are the same, but a good approximation for Min-Uncut is not a good approximation for Min-2-CC.

Thus the cut term is completely determined by the cluster sizes, and not the content of the clusters, so the following remark holds.

Remark 5. *For a given metric graph G, an optimal solution $\mathcal{C}$ to* Min-k-Uncut *on G is an optimal solution to* Min-k-CC *on G if and only if there is an optimal solution to* Min-k-CC $\mathcal{C}'$ *with the same cluster sizes as $\mathcal{C}$.*

Proof. Clearly if there is no optimal solution to Min-k-CC with the same cluster sizes as $\mathcal{C}$, $\mathcal{C}$ cannot be such an optimum. Suppose that there is an optimal solution $\mathcal{C}'$ with the same cluster sizes. Then $e_c(\mathcal{C}) = e_c(\mathcal{C}')$, and, by (5.4),

$$\begin{aligned}
\text{Min-}k\text{-CC}(\mathcal{C}) &= 2\,\text{Min-}k\text{-Uncut}(\mathcal{C}) + e_c(\mathcal{C}) - W \\
&\leq 2\,\text{Min-}k\text{-Uncut}(\mathcal{C}') + e_c(\mathcal{C}') - W \quad \text{as } \mathcal{C} \text{ is } k\text{-Cut optimal,} \\
&= \text{Min-}k\text{-CC}(\mathcal{C}')\,.
\end{aligned}$$

□

Remark 5 has a few immediate implications. Firstly, note that there are certainly cases where the optima are different for the same graph G. For example, consider Figure 5.1(a). In this simple example, the difference between the problems becomes apparent. There is a single set of edges, from x to the points in Z, of length $1/n$. Min-Uncut, desperate to cut edges, will separate x from Z, with cost 0, whereas

MIN-2-CC will respect the fact that the edge weights are small, incurring cost 1. Note that the MIN-UNCUT optimal solution has cost $n-1$ for MIN-2-CC.

Also, it can be proved—see Lemma 15 below—that if a solution $\mathcal{D}$ has the same cluster sizes as some clustering $\mathcal{C}$, and is a good approximation to $\mathcal{C}$ in terms of MIN-k-UNCUT cost, then $\mathcal{D}$ will be a good approximation to $\mathcal{C}$ in terms of MIN-k-CC cost. However, if the cluster sizes are not *exactly* correct, things can go horribly wrong, as the example in Figure 5.1(b) demonstrates. Here, the optimal solution to both problems is given by the clustering $\mathcal{C} = \{\{y\}, \{x\} \cup Z\}$, which has cost $1-n\epsilon$ in the MIN-UNCUT case, and $2-n\epsilon$ in the MIN-2-CC case. Consider the clustering $\mathcal{D} = \{\{x, y\}, Z\}$. It has cost 1 in the MIN-UNCUT case, and thus is a very good approximation (for ϵ small). However, the MIN-2-CC cost of $\mathcal{D}$ is $n+1+n\epsilon$, which is far greater than the MIN-2-CC cost of $\mathcal{C}$. So $\mathcal{D}$ is good approximation for $\mathcal{C}$ in the MIN-UNCUT case, but is not even close for the MIN-2-CC case, even though the cluster sizes differ only slightly from the optimum.

What does all this mean for Indyk's algorithm? Suppose that $\mathcal{C}$ is the optimal MIN-k-CC solution. Then, again assuming we guess the cluster sizes and relevant representatives of $\mathcal{C}$, we can find a clustering $\mathcal{D}$ that is a good MIN-k-UNCUT approximation to $\mathcal{C}$. But are the cluster sizes the same? Can we use Remark 5 and Lemma 15? Let us consider the cases as stated above.

1. **Clusters close.** Here Indyk uses the metric MAX-CUTPTAS. It turns out (see Lemma 15) that a MAX-CC PTAS would also work. We will supply such a PTAS in Section 5.3.

2. **Clusters far, very different sizes.** As UNBALANCED CUT gets cluster sizes exactly right, then as we stated above, Lemma 15 will mean that $\mathcal{D}$ is a good MIN-CC approximation.

3. **Clusters far, similar size.** Unfortunately BALANCED CUT does not get cluster sizes exactly right. However, we can apply the following procedure to *rebalance* the clusters:

Algorithm: RE-BALANCE

Guess the number of points incorrectly placed in cluster D_1,

$$f_{21} = |F_{21}| \qquad F_{21} = D_1 \cap C_2 \tag{5.5}$$

Let G_{21} be the f_{21} points in D_1 which minimise $\hat{\delta}_2$. Define F_{12}, f_{12}, G_{12} similarly.

Output $\mathcal{D}^2$, where

$$D_1^2 = D_1 + G_{12} - G_{21} \qquad D_2^2 = D_2 + G_{21} - G_{12}$$

Our guessing of the correct sizes for each G_{ij} (which approximate F_{ij}) means that, clearly, $|D_i^2| = |C_i|$. We can show (see Lemma 5) that $\mathcal{D}^2$ is a good MIN-UNCUT approximation to $\mathcal{C}$. Again, we are finished, using the all important Lemma 15.

So we can see, modulo a PTAS for metric-MAX-k-CC, which we will supply presently, we have a PTAS for the $k = 2$ case of metric-MIN-k-CC. In Section 5.4, we will outline an algorithm that expands this to the full MIN-k-CC case.

5.3 A PTAS for Dense MAX-k-CC

Let us pause for a moment and consider a slightly wider set of MAX-k-CC problems. We will drop the metric (triangle inequality) requirement, but retain the need for probability constraints. For the MAX-k-CC problem, let us call a graph instance *dense* if the number of edges is in $\Omega(n^2)$. Since the expected value of a random solution to MAX-k-CC, with probability constraints, using only 2 clusters, is $|E|/2$, we know that MAX-k-CC* is in $\Omega(n^2)$ in dense instances. Note that for metric problems $|E|$ is implicitly $\binom{n}{2}$.

Consider again (5.1a) and (5.1b), and the reliance on cluster size. However, if we can define MAX-k-CUT with negative weights, we can then represent the MAX-k-CC cost of a clustering $\mathcal{C}$ in terms of MAX-k-CUT in a *clustering independent* way.

Remark 6. *Let $G = (V, E, w)$ be a weighted graph. Define $\hat{G} = (V, E, \hat{w})$, where $\hat{w} = 2w - 1$. Then, for any k-clustering $\mathcal{C}$:*

$$\textsc{Max-}k\textsc{-CC}(G, \mathcal{C}) = \textsc{Max-}k\textsc{-Cut}(\hat{G}, \mathcal{C}) + |E| - W$$

Proof.

$$\begin{aligned}
\textsc{Max-}k\textsc{-CC}(G, \mathcal{C}) &= \textsc{Max-}k\textsc{-Cut}(G, \mathcal{C}) + e_u(\mathcal{C}) - \textsc{Min-}k\textsc{-Uncut}(G, \mathcal{C}) \\
&= \textsc{Max-}k\textsc{-Cut}(G, \mathcal{C}) + \big[|E| - e_c(\mathcal{C})\big] \\
&\quad + \big[\textsc{Max-}k\textsc{-Cut}(G, \mathcal{C}) - W)\big] \\
&= 2\,\textsc{Max-}k\textsc{-Cut}(G, \mathcal{C}) - e_c(\mathcal{C}) + |E| - W \\
&= \sum_{e \in E_c(\mathcal{C})} (2w(e) - 1) + |E| - W \\
&= \textsc{Max-}k\textsc{-Cut}(\hat{G}, \mathcal{C}) + |E| - W
\end{aligned}$$

□

Remark 6 tells us, for instance, that the optimal solution to Max-k-Cut on $\hat{G}$ is the same as the optimal solution of Max-k-CC on G. This observation is key to our applying the following theorem to our problem.

Theorem 3 (Frieze and Kannan [49], Theorem 1)**.** *There is an algorithm, which we refer to as* FK*, that given a graph $\hat{G} = (V, E, \hat{w})$, with weight function $\hat{w} : E \to [-1, 1]$, and a fixed $\epsilon > 0$, computes in polynomial time a k-clustering $\mathcal{C}$ such that:*

$$\textsc{Max-}k\textsc{-Cut}(\mathcal{C}) \geq \textsc{Max-}k\textsc{-Cut}^* - \epsilon n^2$$

Corollary 5. *If $|E| \in \Omega(n^2)$, then algorithm* FK *provides a PTAS for* Max-k-CC.

Proof. Let $\mathcal{C}$ be the solution returned by algorithm FK, run on $\hat{G}$, and $\mathcal{C}^*$ be the optimal solution to Max-k-CC. Then, by Remark 6,

$$\begin{aligned}
\textsc{Max-}k\textsc{-CC}^* - \textsc{Max-}k\textsc{-CC}(\mathcal{C}) &= \textsc{Max-}k\textsc{-Cut}(\hat{G}, \mathcal{C}^*) - \textsc{Max-}k\textsc{-Cut}(\hat{G}, \mathcal{C}) \\
&\leq \textsc{Max-}k\textsc{-Cut}(\hat{G})^* - \big[\textsc{Max-}k\textsc{-Cut}(\hat{G})^* - \epsilon n^2\big] \\
&= \epsilon n^2
\end{aligned}$$

and we are done, since MAX-k-CC* is in $\Omega(n^2)$. □

Frieze and Kannan's algorithm runs in time $\alpha(\epsilon)M(n)\beta(\epsilon)$, where $\alpha(\epsilon) = O(\epsilon^{-20})$, $M(n)$ is the time to multiply two $n \times n$ 0/1 matrices, and $\log\log\beta(\epsilon) \leq \frac{c_1 \log(1/\epsilon)}{\epsilon^8}$, for some constant c_1.

5.4 A PTAS for Metric MIN-k-CC

In this section, we will outline the PTAS for the metric-MIN-k-CC problem. We will begin by outlining the operation of the PTAS for metric-MIN-k-UNCUT of Fernandez de la Vega et al. [43], as our PTAS expands that algorithm, and requires much of the same terminology. We will also report the relevant sections of their analysis which we will use to analyse our algorithm.

5.4.1 The PTAS of Fernandez de la Vega et al.

We call the algorithm of Fernandez de la Vega et al. the FKKR algorithm. This algorithm is a generalisation of Indyk's algorithm, which we described above, to $k > 2$ clusters. The underlying idea, of exploiting *well-separatedness* of clusters, remains, however, things are more difficult when there are more than two clusters. In this section we will be focused exclusively on the MIN-k-UNCUT objective.

Indyk's algorithm had three scenarios to deal with. The three scenarios depended on the configuration of the optimal solution for the MIN-k-UNCUT problem, which, in this section we'll term $\mathcal{C}$.[1] The scenarios that Indyk outlined depended on two qualities of the clusters in $\mathcal{C}$; the *closeness* of the clusters, and the *size* of the clusters. Let us be precise about what these mean.

Cluster Size. Let $n_i = |C_i|$, and let us assume without loss of generality that $n_1 \geq n_2 \geq \cdots \geq n_k$. Since we are describing a PTAS, we have been given an ϵ. Indyk only needed to know if $n_1 \geq \epsilon\, n_2$, however in the general case a very technical definition of large and small is required. Let $I_j = (\epsilon^{j+1}, \epsilon^j]$, and let $j_0 < k^2$ be the

[1] In the analysis of our algorithm, $\mathcal{C}$ is the optimal solution to the MIN-k-CC problem. Here it is the optimal solution to MIN-k-UNCUT. We use the same symbol, as our algorithm treats the MIN-k-CC optimum much the same as FKKR treats the MIN-k-UNCUT optimum.

minimum j such that for every i, i', the ratio $n_i/n_{i'} \notin I_j$.Let $M = n_1$, be the size of the largest cluster. Let $k_0 = \text{argmin}_i\, n_i \geq \epsilon^{j_0} n_1$.

Definition 18. *A cluster C_i is* large *if $i \leq k_0$ and* small *otherwise.*

The set S is the union of the small clusters: $\cup_{i>k_0} C_i$. Finally, $m = n_{k_0}$, the size of the smallest large cluster, while $s = n_{k_0+1}$, the size of the largest small cluster. We can see by the above definition that $s = O(\epsilon)m$.

Representatives Large clusters are of particular interest. Firstly, they contribute the majority of the cost of $\mathcal{C}$, and thus approximating them well is of prime importance. Secondly, it is easy to find representatives of them:

Lemma 9 (F. de la Vega et al., Lemma 5)**.** *For each $i \leq k_0$, let c_i be chosen uniformly at random, and independently, from V. Then with probability at least $[\epsilon^{j_0}/(2k)]^k$: each c_i chosen is a representative of C_i.*

Representatives are useful because we can use a representative c_i to estimate the distance from any point to the optimal cluster C_i. Assuming $\mathcal{C}$ is fixed, let us define, for $i \leq k_0$,

Definition 19.

$$\delta_i(v) = \delta(v, C_i) \qquad \hat{\delta}_i(v) = n_i\, \delta(v, c_i) \tag{5.6}$$

Then Lemma 8 tells us that, up to a small error, $\hat{\delta}_i$ is a good stand-in for δ_i. Or, precisely

$$\left|\hat{\delta}_i(v) - \delta_i(v)\right| \leq \frac{2}{m}\, \text{MIN-}k\text{-UNCUT}(\mathcal{C})\,. \tag{5.7}$$

Separating Large Clusters From Small

Indyk demonstrated that when one cluster was large and the other small, the UNBALANCED CUT algorithm was sufficient to solve the problem. That is, we can easily find a representative c_i for the large cluster C_i, and then choose the closest n_i points to c_i. This works because it does not matter significantly which points end up the small cluster, and (5.7) tells us that using the representative is sufficiently accurate.

In the full FKKR algorithm, Fernandez de la Vega et al. use the same idea in order to separate out the small clusters in Step 5.4.1.

Separating Two Large Clusters

When separating two large clusters, C_i and C_j, what is important is whether the clusters were *well-separated*. We can make this explicit with the concept of *close* clusters. We say two large clusters are close if

$$\delta(C_i, C_j) \leq \beta\big[\delta(C_i) + \delta(C_j)\big]\,, \tag{5.8}$$

where $\beta = M/(m\epsilon)$.

If two clusters are close, then a metric-Max-k-Cut PTAS can separate them well. This is due to the following fundamental fact about maximisation/minimisation:

Remark 7. *Let* Max-X *and* Min-Y *be complementary problems; that is, for any graph G,* Max-X$(\mathcal{X})$ + Min-Y$(\mathcal{X})$ *is constant over all solutions $\mathcal{X}$. If*

$$f(\epsilon)\,\text{Max-X}^* \leq \text{Min-Y}^*\,, \tag{5.9}$$

then a $(1-g(\epsilon))$-approximate solution to Max-X *will be a $(1+g(\epsilon)/f(\epsilon))$-approximate solution to* Min-Y*.*

Proof. Let $\mathcal{X}$ be such that Max-X$(\mathcal{X}) \geq (1-g(\epsilon))$Max-X*. Then, as Max-X and Min-Y are complementary, we know

$$\text{Max-X}(\mathcal{X}) + \text{Min-Y}(\mathcal{X}) = \text{Max-X}^* + \text{Min-Y}^*\,.$$

So,

$$\begin{aligned}\text{Min-Y}(\mathcal{X}) &= \text{Min-Y}^* + \text{Max-X}^* - \text{Max-X}(\mathcal{X}) \\ &\leq \text{Min-Y}^* + g(\epsilon)\text{Max-X}^* \\ &\leq \text{Min-Y}^* + (g(\epsilon)/f(\epsilon))\,\text{Min-Y}^*\end{aligned}$$

□

This leads in a straightforward way to a comment about PTASes:

Corollary 6. *Let* MAX-X *and* MIN-Y *be complementary problems, with*

$$f(\epsilon)\,\text{MAX-X}^* \leq \text{MIN-Y}^*\,,$$

Then if, for any ϵ, $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation algorithm for MAX-X*, then for any ϵ, $\mathcal{A}_{\epsilon f(\epsilon)}$ is a $(1+\epsilon)$-approximation algorithm for* MIN-Y*.*

So, if two clusters are close, (5.8) provides an example of (5.9), letting us apply Corollary 6 to the metric-MAX-k-CUT PTAS.

Indyk had demonstrated that BALANCED CUT is sufficient to separate two large clusters that are not close. This is due to the fact that such clusters have a good separation; so using representatives as an approximation to cluster distance—using (5.7)—does a good enough job. In the multiple cluster case, however, there is an added difficulty.

Groups of Large Clusters

Suppose clusters C_x and C_y are close, as are C_y and C_z. However, suppose C_x and C_z are not close. We could separate C_x and C_z via the BALANCED CUT algorithm. However, the addition of C_y will confuse this algorithm. In fact, it turns out that a metric-MAX-k-CUT PTAS will be able to separate these three. We need only use BALANCED CUT to separate clusters that are not only not close, but also not connected by a series of close clusters.

To make this explicit, we need the concept of large cluster groups. Let us define an equivalence relationship $\equiv$ on the cluster indices, so that $i \equiv i'$ if there is a sequence of indices $x_1 = i, x_2, \ldots, x_\ell = i'$, such that C_{x_z} and $C_{x_{z+1}}$ are close for all $z < \ell$. We can use this equivalence relationship to find a *group mapping function* $g : [k] \to \{0, 1, \ldots, \gamma\}$, where γ is the number of equivalence classes, and $g(i) = g(i')$ if $i \equiv i'$. We let $g(i) = 0$ for all $i > k_0$, and define $m_j = |g^{-1}(j)|$ for $j \in [\gamma]_0$. For any clustering $\mathcal{X}$, let the *restriction* of $\mathcal{X}$ to group j, $\mathcal{X}|_j$ be the (clustering) function $\mathcal{X}$ on the domain $\{v : g(\mathcal{X}(v)) = j\}$.

It is important to know when two clusterings make the same decisions about groups, but perhaps differ within them.

Definition 20. *We say two clusterings $\mathcal{X}$ and $\mathcal{Y}$ are* g-equivalent *if for all $v \in V$,*

$$g(\mathcal{X}(v)) = g(\mathcal{Y}(v))\,.$$

The following lemma, proved by Fernandez de la Vega et al., demonstrates that a group can be approximated by a metric-Max-k-Cut PTAS, with an application of Remark 7:

Lemma 10 (F. de la Vega et al.)**.** *Let $j \in [\gamma]$ be a group of large clusters. Then*

$$O(\epsilon)\text{Max-}m_j\text{-Cut}(\mathcal{C}|_j) \leq \text{Min-}m_j\text{-Uncut}(\mathcal{C}|_j)$$

The FKKR algorithm

We are now in a position to detail the FKKR algorithm. The algorithm operates in 4 phases:

Algorithm: FKKR

Phase Zero: Guess the values of all n_i, $i \in [k]$, and the group mapping function g, and guess the c_i, $i \in [k_0]$, as in Lemma 9.

Phase One: Define the clustering $\mathcal{D}^1 : V \to [k_0]$ as follows:

$$\mathcal{D}^1(v) = \operatorname*{argmin}_{i \leq k_0} \hat{\delta}_i(v)\,.$$

Define the (unknown) $F_{ii'}$ to be $C_i \cap D^1_{i'}$, and let $f_{ii'}$ stand for $|F_{ii'}|$.

Phase Two: Guess $f_{ii'}$ for all $i > k_0$ small, and let $f(v) = \min_{i \equiv \mathcal{C}(v)} \delta(c_i, v)$. For each large group $j \in [\gamma]$, remove the $\sum_{\substack{i:g(i)=j \\ i':g(i')=0}} f_{i'i}$ elements with smallest f values from clusters in j, and place them in arbitrary small clusters.

Phase Three: For each group $j \in [\gamma]_0$, let $G^2_j = \{v : g(\mathcal{D}^2(v)) = j\}$. For $j \in [\gamma]$, partition G^2_j into m_j clusters using a dense-Max-k-Cut PTAS. Partition G_0 into $k - k_0$ clusters, by (recursively) applying the algorithm in this section.

This algorithm runs in time $O(f(k,\epsilon)n^{3k})$, where $f(k,\epsilon) = O(\exp((1/\epsilon)^{k^2}))$.

Analysis of the FKKR Algorithm

We provide some important elements of Fernandez de la Vega et al.'s analysis, as they will prove useful in our own work.

Consider a clustering $\mathcal{C}^1$ that is a compromise between $\mathcal{D}^1$ and $\mathcal{C}$. Its small clusters are the same as those in $\mathcal{C}$. Its large clusters are the same as those in $\mathcal{D}^1$, *except* that the points in S have been removed, and within each group, points are clustered as they are in $\mathcal{C}$.

That is,

$$\mathcal{C}^1(v) = \begin{cases} \mathcal{C}(v) & \text{if } \mathcal{D}^1(v) \equiv \mathcal{C}(v) \text{ or } g(\mathcal{C}(v)) = 0\,, \\ \mathcal{D}^1(v) & \text{otherwise}\,. \end{cases}$$

Fernandez de la Vega et al. show the following results:

Proposition 4 (F. de la Vega et al. Prop. 9)**.**

$$\text{MIN-}k\text{-UNCUT}(\mathcal{C}^1) \leq \big(1 + O(\epsilon)\big)\,\text{MIN-}k\text{-UNCUT}(\mathcal{C})\,.$$

Lemma 11 (F. de la Vega et al. Lemma 11)**.** *For $i, i' \leq k_0$ and $i' \not\equiv i$, $|F_{ii'}| \leq O(\epsilon)m$.*

Theorem 4 (F. de la Vega et al.)**.** *The algorithm* FKKR *is PTAS for the metric-*MIN-k-UNCUT *problem.*

5.4.2 Our Algorithm

We now assume that $\mathcal{C}$ is an optimal solution to the MIN-k-CC problem on some graph G. So MIN-k-CC* = MIN-k-CC($\mathcal{C}$). We will be interested in the MIN-k-UNCUT cost of $\mathcal{C}$—in fact we will use part of the FKKR algorithm to approximate it—so let the symbol c^* stands for $\sum_i \delta(C_i) = 2\,\text{MIN-}k\text{-UNCUT}(\mathcal{C})$.

Our algorithm operates in a number of phases, similar to the FKKR algorithm.

Phase Zero: Guess the values of all n_i, $i \in [k]$, and the group mapping function g, and guess the c_i, $i \in [k_0]$, as in Lemma 9.

Phase One: Define the clustering $\mathcal{D}^1 : V \to [k_0]$ as follows:
Let $\mathcal{D}^1(v) = \text{argmin}_{i \leq k_0} \hat{\delta}_i(v)$. Define the (unknown) $F_{ii'}$ to be $C_i \cap D^1_{i'}$, and let $f_{ii'}$ stand for $|F_{ii'}|$.

Phase Two: Define the clustering $\mathcal{D}^2$, by solving a BIPARTITE MATCHING problem:

1. Guess all $f_{ii'}$ for each $i \in [k]$ and each $i' \in [k_0]$.
2. Define a weighted bipartite graph as follows: Let $L = V$, and

$$R = \bigcup_{i \in [k], i' \in [k_0]} \{f_{ii'} \text{ vertices, each labelled } G_{ii'}\}$$

   $\tilde{E}$ consists of an edge between each $x \in L = V$, and each vertex in R labelled $G_{i\mathcal{D}^1(x)}$, for each $i \in [k]$, with weight $\tilde{w}(e) = \tilde{\delta}_i(x)$.
3. Solve BIPARTITE MATCHING, on $\tilde{G}(L \cup R, \tilde{E}, \tilde{w})$ to get a perfect matching. Let $\mathcal{D}^2(x) = i$ if $x \in L$ is matched to an element labelled $G_{i\mathcal{D}^1(x)}$.

Phase Three: For each group $j \in [\gamma]_0$, let $G^2_j = \{v : g(\mathcal{D}^2(v)) = j\}$. For $j \in [\gamma]$, partition G^2_j into m_j clusters using the MAX-k-CC PTAS of Section 5.3. Partition G_0 into $k - k_0$ clusters, by (recursively) applying the algorithm in this section.

5.4.3 The Analysis

We analyse the progress by comparing the clusterings actually obtained with some nearer-optimal, though unknown, clusterings called $\mathcal{C}^1$ and $\mathcal{C}^2$.

Balanced Cut: $\mathcal{D}^1$ and $\mathcal{C}^1$

Phases Zero and One are identical to those phases from FKKR. Thus, if we form $\mathcal{C}^1$ as we did in Section 5.4.1, we can follow Fernandez de la Vega et al.'s analysis through[2] and find that Proposition 4 and Lemma 11 hold, for this particular $\mathcal{C}$.

[2]Note that Fernandez de la Vega et al.'s analysis does not rely in any way on $\mathcal{C}$ being the optimal solution to the MIN-k-UNCUT problem.

Re-balancing: $\mathcal{D}^2$ and $\mathcal{C}^2$

Phase Two serves two purposes. First, we separate the points that we believe should go in small clusters from those which we believe should be in large clusters. Second, we force the clusters to have the same sizes as the (guessed) optimum solution sizes $\{n_i\}$. This algorithm relies on techniques for the MIN-k-UNCUT problem to provide a good solution to the MIN-k-CC problem. Referring back to Remark 5, if the clusters sizes are the same, then optimality of one problem implies optimality of the other.

By the construction of Phase Two, the clusters in $\mathcal{D}^2$ are the correct sizes. We need to show that there exists a clustering, $\mathcal{C}^2$, that also has correct sizes, but is a good MIN-k-UNCUT approximation to $\mathcal{C}$. Define $G_{ii'}$ to be $D^2_i \cap D^1_{i'}$. The elements of $G_{ii'}$ are the vertices that move from cluster i' to cluster i in Phase Two. By definition, $|G_{ii'}| = f_{ii'} = |F_{ii'}|$. Indeed, $G_{ii'}$ is our attempt at finding $F_{ii'}$; if we had $G_{ii'} = F_{ii'}$ for all i, i', we would have $\mathcal{D}^2 = \mathcal{C}$.

Pairing functions For each (i, i'), the sets $F_{ii'}$ and $G_{ii'}$ are of the same size. We can therefore find a pairing function p (a bijection) between them. There are many choices of such pairing functions, but we will choose one that has the following property.

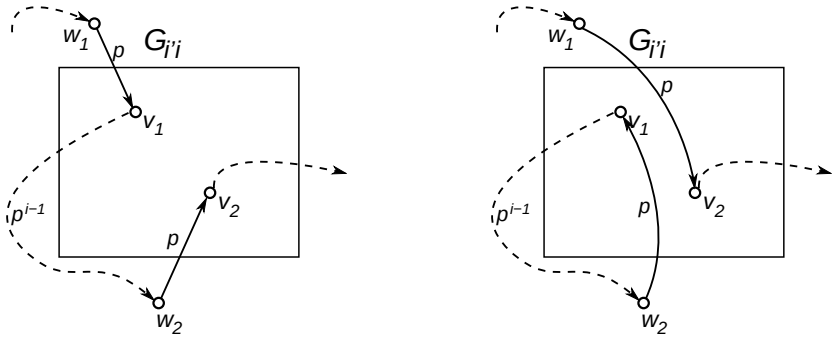
Definition 21. *A pairing function p has* small-loops*, if, for each $v \in C_i$, the orbit of v (the set of points reachable from v by repeated application of p) enters each $G_{ii'}$ at most once for each $i \in [k]$.*

Remark 8. *There exists a pairing function p with small-loops.*

Proof. It is simple to find such a pairing function; suppose we have a pairing function p such that $v_1, v_2 \in G_{i'i}$ are in the same orbit. That is, $p^x(v_1) = v_2$ for some x. Then, as v_1, v_2 are both in $G_{i'i}$, there are two nodes $w_1, w_2 \in F_{i'i}$ such that $p(w_1) = v_1$ and $p(w_2) = v_2$. Then we can define a new pairing function p' such that $p'(w_2) = v_1$, $p'(w_1) = v_2$, and $p'(w) = p(w)$ otherwise.

The procedure is demonstrated in Figure 5.2.

After this change, p' is still a pairing function, and v_1 and v_2 are now on separate orbits (v_1 on a loop of length x). Repeated application of the above procedure



(a) Pairing function p: v_1 and v_2 are on the same orbit, yet both in $G_{i'i}$.

(b) Pairing function p': v_1 and v_2 are now on two distinct orbits.

Figure 5.2: Changing a pairing function p to p' in order to make smaller loops.

will lead to a pairing function with small-loops. Since points that were not on the same orbit before the procedure are not on the same orbit after the procedure, the repeated application must terminate. □

Some properties of small-loop pairing functions p are immediate. Each vertex $v \in F_{ii'} \cap G_{ii'}$ has $p(v) = v$; that is, each value $\mathcal{D}^2(v)$ is unique in a given orbit. Consequently, every orbits must be of length at most k. Note also, from the definition, that an orbit is a subset of some D_i^1, all $\mathcal{D}^1(v)$ values are the same, and that $\mathcal{D}^2(p(v)) = \mathcal{C}(v)$, a fact that will be used often.

Definition 22. *An orbit o is* group-contained *if for all $v \in o$, $\mathcal{C}(v) \equiv \mathcal{D}^1(v)$.*

By definition, $\mathcal{C}^1(v) = \mathcal{C}(v)$ for all v in a group-contained orbit, and it is not hard to show that $\mathcal{D}^2(v) \equiv C(v)$ for all v in the orbit too. Let us denote the vertices *not* in group-contained orbits by V_1.

We are now in a position to define $\mathcal{C}^2$ precisely:

$$\mathcal{C}^2(v) = \begin{cases} \mathcal{D}^2(v) & \text{if } v \in V_1, \\ \mathcal{C}(v) & \text{otherwise.} \end{cases}$$

Remark 9. *The size of the set V_1 is in $O(\epsilon m)$.*

Proof. By definition, each non-group contained orbit has some vertex with $\mathcal{D}^1(v) \not\equiv \mathcal{C}(v)$, that is, in $F_{ii'}$ with $i \not\equiv i'$. From Lemma 11, we know that $\sum_{i \not\equiv i'} |F_{ii'}| \in$

$O(k^2\epsilon m)$, if $i, i' \leq k_0$. In addition, if $i > k_0$, then $|F_{ii'}| \leq |C_i|$, as it is a subset, and this is less than m. Each (small-loop) orbit has at most k vertices, which proves the Remark. □

Remark 10. *$\mathcal{C}^2$ is g-equivalent to $\mathcal{D}^2$.*

Proof. Clearly points in V_1 are g-equivalent. If $v \in V_0$, then we know by definition that

$$\mathcal{D}^2(v) = \mathcal{C}(p^{-1}(v)) \equiv \mathcal{D}^1(p^{-1}(v)) = \mathcal{D}^1(v) \equiv \mathcal{C}(v) = \mathcal{C}^2(v)\,.$$

□

Remark 11. *Without loss of generality, if $g(\mathcal{C}(v)) = g(\mathcal{D}^2(v)) = 0$, then $\mathcal{C}(v) = \mathcal{D}^2(v)$.*

Proof. Suppose this is not the case for some v satisfying $g(\mathcal{D}^2(v)) = g(\mathcal{C}(v)) = 0$, but not $\mathcal{D}^2(v) = \mathcal{C}(v)$. We will define a new optimal solution to the BIPARTITE MATCHING problem, $\mathcal{D}^{2'}$, which is g-equivalent to $\mathcal{D}^2$. Repeating in this fashion for all v failing the condition will lead to a clustering that satisfies Remark 11. Define $\mathcal{D}^{2'}$ identically to $\mathcal{D}^2$, except let $D^{2'}(v) = C(v)$, which is of course $D^2(p^{-1}(v))$, and $D^{2'}(p^{-1}(v)) = D^2(v)$. Thus v and $p^{-1}(v)$ are exchanging roles. We can then reduce p to regain the small-loops property. As $g(\mathcal{D}^2(v)) = g(\mathcal{D}^{2'}(v)) = 0$ the assignment cost in the BIPARTITE MATCHING is unchanged, so $\mathcal{D}^{2'}(v)$ is an optimal solution. Moreover, $\mathcal{D}^{2'}$ is g-equivalent to $\mathcal{D}^2$, and has clusters of the same sizes as $\mathcal{D}^2$. So none of the later of the analysis will be affected, and we assume $\mathcal{D}^2$ has the required property. □

Corollary 7. *No orbit has two consecutive nodes mapped to small clusters by $\mathcal{C}$.*

Proof. Suppose that $g(\mathcal{C}(p^{-1}(v))) = g(\mathcal{C}(v)) = 0$ for some $v \neq p(v)$. This would imply that $g(\mathcal{D}^2(v)) = 0$ also, and therefore from Remark 11, that $\mathcal{D}^2(v) = \mathcal{C}(v)$. Finally, this implies $\mathcal{D}^2(v) = \mathcal{D}^2(p(v))$, which breaks the small loops property. □

The Cost of $\mathcal{C}^2$

We will now state a vital Lemma, which is a simple consequence of the fact that $\mathcal{D}^2$ was the result of an optimal solution to the BIPARTITE MATCHING problem we set up in Phase Two.

Lemma 12.

$$\sum_{v \in V_1} \tilde{\delta}_{\mathcal{D}^2(v)}(v) \leq \sum_{v \in V_1} \tilde{\delta}_{\mathcal{C}(v)}(v)$$

Proof. To prove this, we appeal to the optimality of $\mathcal{D}^2$ as a solution to the Bipartite Matching problem on $\tilde{G}$. Consider another solution to Bipartite Matching, μ, defined as $\mu(v) = \mathcal{C}(v)$ for $v \in V_1$ and $\mu(v) = \mathcal{D}^2(v)$ otherwise. This function indeed provides a matching, as $\mathcal{C}(v) = \mathcal{D}^2(p(v))$ for any v, and no orbit enters or leaves V_1. So, considering the costs of the two matchings, as $\mathcal{D}^2$ is optimal, and only differs from μ on V_1, the lemma follows. □

We can use this fact, along with properties of the choice made in Phase One, to show that the vertices that we mistakenly move in Phase Two are close to the vertices we should have moved.

Lemma 13.

$$\sum_{v \in V_1} \delta(v, p(v)) \leq O(1)\frac{c^*}{m}$$

Proof. If v is in a orbit of length one, $p(v) = v$, and v contributes 0 to the above sum.
Otherwise, recall that $\mathcal{D}^1(v) = \mathcal{D}^1(p(v))$, so an application of the triangle inequality and the definition of $\hat{\delta}$ gives

$$\delta(v, p(v)) \leq \frac{1}{n_{\mathcal{D}^1(v)}} \left[\hat{\delta}_{\mathcal{D}^1(v)}(v) + \hat{\delta}_{\mathcal{D}^1(p(v))}(p(v))\right]$$

Since $p(v) \neq v$, $\mathcal{C}(v) = \mathcal{D}^2(p(v)) \neq \mathcal{D}^2(v)$, so we apply the contrapositive of Remark 11 to learn that either $\mathcal{C}(v)$ or $\mathcal{D}^2(v)$ is in $[k_0]$.

Now, the choice of the algorithm in Phase One means that $\hat{\delta}_{\mathcal{D}^1(v)}(v) \leq \hat{\delta}_i(v)$ for every $i \in [k_0]$. This means that we can bound $\hat{\delta}_{\mathcal{D}^1(v)}(v)$ above by $\tilde{\delta}_{\mathcal{C}(v)}(v) + \tilde{\delta}_{\mathcal{D}^2(v)}(v)$, as at least one of these terms is actually a $\hat{\delta}$. Therefore,

$$\sum_{v \in V_1} \delta(v, p(v)) \leq \frac{1}{m} \sum_{v \in V_1} \left[\tilde{\delta}_{\mathcal{C}(v)}(v) + \tilde{\delta}_{\mathcal{D}^2(v)}(v) + \tilde{\delta}_{\mathcal{C}(p(v))}(p(v)) + \tilde{\delta}_{\mathcal{D}^2(p(v))}(p(v))\,.\right]$$

Recall that $v \in V_1 \Rightarrow p(v) \in V_1$, so we have an upper bound of

$$\sum_{v \in V_1} \delta(v, p(v)) \leq \frac{2}{m} \sum_{v \in V_1} \left[\tilde{\delta}_{\mathcal{C}(v)}(v) + \tilde{\delta}_{\mathcal{D}^2(v)}(v) \right] .$$

From Lemma 12, we know that this is at most

$$\frac{4}{m} \sum_{v \in V_1} \tilde{\delta}_{\mathcal{C}(v)}(v) = \frac{4}{m} \sum_{v \in V_1, g(\mathcal{C}(v)) \neq 0} \hat{\delta}_{\mathcal{C}(v)}(v) \leq \frac{4}{m} \left[\sum_{v \in V_1, g(\mathcal{C}(v)) \neq 0} \delta_{C(v)}(v) + |V_1| \frac{2c^*}{m} \right] ,$$

by Lemma 8. Recalling Remark 9, and the fact that $c^* = \sum_{v \in V} \delta_{\mathcal{C}(v)}(v)$, the proof is complete. □

Definition 23. *For each i, let $In_i = C_i^2 \setminus C_i$, and $Out_i = C_i \setminus C_i^2$, so that $C_i^2 = C_i + In_i - Out_i$.*

Remark 12. *$Out_i = \{p^{-1}(v) \mid v \in In_i\}$, and so $|C_i^2| = |C_i|$.*

Proof. Let $X = \{p^{-1}(v) \mid v \in \mathrm{In}_i\}$. If $v \in \mathrm{In}_i$, then $\mathcal{C}^2(v) = i \neq \mathcal{C}(v)$ and v is not in a group-contained orbit. Let $u = p^{-1}(v)$, then $u \in X$, and $\mathcal{C}(u) = \mathcal{D}^2(p(u)) = \mathcal{D}^2(v) = \mathcal{C}^2(v) = i$, which also implies $u \neq v$. On the other hand, $\mathcal{C}^2(u) \neq \mathcal{C}^2(v) = i$, by the small-loops property, so $u \in \mathrm{Out}_i$. Consequently, $X \subseteq \mathrm{Out}_i$. A similar argument in the other direction shows that $\mathrm{Out}_i \subseteq X$. □

Lemma 14.

$$\delta(C_i^2) - \delta(C_i) \leq 2 \left[\delta(C_i, In_i) - \delta(C_i, Out_i) \right] + O(\epsilon) c^* .$$

Proof. Expanding $\delta(C_i^2)$ gives

$$\begin{aligned} \delta(C_i^2) = \delta(C_i) + 2 \left[\delta(C_i, \mathrm{In}_i) - \delta(C_i, \mathrm{Out}_i) \right] + \\ \left[\delta(\mathrm{In}_i) - \delta(\mathrm{In}_i, \mathrm{Out}_i) \right] + \left[\delta(\mathrm{Out}_i) - \delta(\mathrm{Out}_i, \mathrm{In}_i) \right] \end{aligned}$$

Consider the second-last term,

$$\begin{aligned}
\delta(\mathrm{In}_i) - \delta(\mathrm{In}_i, \mathrm{Out}_i) &= \sum_{u \in \mathrm{In}_i} [\delta(u, \mathrm{In}_i) - \delta(u, \mathrm{Out}_i)] \\
&= \sum_{u \in \mathrm{In}_i} \sum_{v \in \mathrm{Out}_i} [\delta(u, p(v)) - \delta(u, v)] \\
&\leq \sum_{u \in \mathrm{In}_i} \sum_{v \in \mathrm{Out}_i} \delta(v, p(v)) && \text{(triangle inequality)} \\
&\leq |V_1| \sum_{v \in V_1} \delta(v, p(v)) && (\mathrm{In}_i, \mathrm{Out}_i \subseteq V_1) \\
&\leq O(\epsilon) c^* \,. && \text{(Remark 9 and Lemma 13)}
\end{aligned}$$

We can bound the final term in the same way, which completes the proof. □

Remark 13.

$$\sum_{i \in [k]} \delta(C_i, In_i) - \delta(C_i, Out_i) = \sum_{v \in V_1} \left[\delta_{\mathcal{C}^2(v)}(v) - \delta_{\mathcal{C}(v)}(v)\right]$$

Proof. From Remark 12,

$$\begin{aligned}
\sum_{i \in [k]} \delta(C_i, \mathrm{In}_i) - \delta(C_i, \mathrm{Out}_i) &= \sum_{i \in [k]} \sum_{v \in \mathrm{In}_i} \left[\delta_i(v) - \delta_i(p^{-1}(v))\right] \\
&= \sum_{i \in [k]} \sum_{v \in \mathrm{In}_i} \left[\delta_{\mathcal{C}^2(v)}(v) - \delta_{\mathcal{C}(p^{-1}(v))}(p^{-1}(v))\right]
\end{aligned}$$

For a vertex u in V_1 that is not in some In_i, $p(u) = u$, so the corresponding term inside the brackets above is zero. Therefore the right hand side is, after separating the terms,

$$\sum_{v \in V_1} \delta_{\mathcal{C}^2(v)}(v) - \sum_{v \in V_1} \delta_{\mathcal{C}(p^{-1}(v))}(p^{-1}(v)) = \sum_{v \in V_1} \delta_{\mathcal{C}^2(v)}(v) - \sum_{v \in V_1} \delta_{\mathcal{C}(v)}(v)\,,$$

as $v \in V_1$ implies $p^{-1}(v) \in V_1$. □

Theorem 5. *$\mathcal{C}^2$ is a $(1+O(\epsilon))$ approximation to $\mathcal{C}$ in terms of* Min-k-Uncut *cost.*

Equivalently,

$$\sum_i \delta(C_i^2) \leq (1 + O(\epsilon))c^* \,.$$

Proof. From Lemma 14 and Remark 13,

$$\sum_i \delta(C_i^2) \leq \sum_i \delta(C_i) + O(\epsilon)c^* + \sum_{v \in V_1} \left[\delta_{\mathcal{C}^2(v)}(v) - \delta_{\mathcal{C}(v)}(v)\right]$$

Now,

$$\begin{aligned}\sum_{\substack{v \in V_1 \\ g(\mathcal{C}^2(v)) \neq 0}} \delta_{\mathcal{C}^2(v)}(v) - \sum_{\substack{v \in V_1 \\ g(\mathcal{C}(v)) \neq 0}} \delta_{\mathcal{C}(v)}(v) \\ \leq \sum_{\substack{v \in V_1 \\ g(\mathcal{C}^2(v)) \neq 0}} \tilde{\delta}_{\mathcal{C}^2(v)}(v) - \sum_{\substack{v \in V_1 \\ g(\mathcal{C}(v)) \neq 0}} \tilde{\delta}_{\mathcal{C}(v)}(v) + |V_1|\frac{4c^*}{m} \in O(\epsilon c^*)\,,\end{aligned}$$

from Lemma 8, Lemma 12, and Remark 9. In contrast,

$$\sum_{\substack{v \in V_1 \\ g(\mathcal{C}^2(v)) = 0}} \delta_{\mathcal{C}^2(v)}(v) - \sum_{\substack{v \in V_1 \\ g(\mathcal{C}(v)) = 0}} \delta_{\mathcal{C}(v)}(v) = \sum_{\substack{v \in V_1 \\ g(\mathcal{C}(v)) = 0}} \delta_{\mathcal{C}(v)}(p(v)) - \sum_{\substack{v \in V_1 \\ g(\mathcal{C}(v)) = 0}} \delta_{\mathcal{C}(v)}(v)$$

Now, Proposition 3 shows that $\delta_{\mathcal{C}(v)}(p(v)) - \delta_{\mathcal{C}(v)}(v) \leq |C_{\mathcal{C}(v)}|\delta(v, p(v))$, so

$$\begin{aligned}\sum_{v \in V_1, g(\mathcal{C}(v)) = 0} \left[\delta_{\mathcal{C}(v)}(p(v)) - \delta_{\mathcal{C}(v)}(v)\right] &\leq \sum_{v \in V_1, g(\mathcal{C}(v)) = 0} |C_{\mathcal{C}(v)}|\delta(v, p(v)) \\ &\leq O(1)\frac{s}{m}c^* + \leq O(\epsilon)c^*\,,\end{aligned}$$

where the final inequality follows from Lemma 13 and the facts that $|C_{\mathcal{C}(v)}| \leq s$ and $s \leq m\epsilon$.

□

Analysing Phase Three.

At the conclusion of Phase Two, we have a clustering $\mathcal{D}^2$ that is g-equivalent to $\mathcal{C}^2$, a $(1 + O(\epsilon))$-approximation to $\mathcal{C}$ in MIN-k-UNCUT terms. Moreover, $\mathcal{D}^2$, $\mathcal{C}^2$ and $\mathcal{C}$ all have clusters of the same sizes. We now show that $\mathcal{D}^3$, the outcome of Phase

Three, is a $(1 + O(\epsilon))$-approximation to $\mathcal{C}$ in Min-k-CC terms.

Lemma 15. *If* $f(\epsilon)$ *satisfies* Min-k-Uncut$(\mathcal{C}^2) \leq (1 + f(\epsilon))$ Min-k-Uncut$(\mathcal{C})$, *then*

$$\text{Min-}k\text{-CC}(\mathcal{C}^2) \leq (1 + 2f(\epsilon))\,\text{Min-}k\text{-CC}(\mathcal{C})\,.$$

Proof. Note that, for any clustering $\mathcal{X}$:

$$\begin{aligned}\text{Min-}k\text{-CC}(\mathcal{X}) &= \text{Min-}k\text{-Uncut}(\mathcal{X}) + e_c(\mathcal{X}) - \text{Max-}k\text{-Cut}(\mathcal{X}) \\ &= 2\,\text{Min-}k\text{-Uncut}(\mathcal{X}) + e_c(\mathcal{X}) - W\end{aligned}$$

Consider

$$\begin{aligned}\text{Min-}k\text{-CC}(\mathcal{C}^2) - \text{Min-}k\text{-CC}(\mathcal{C}) &= 2\left(\text{Min-}k\text{-Uncut}(\mathcal{C}^2) - \text{Min-}k\text{-Uncut}(\mathcal{C})\right) \\ &\qquad + e_c(\mathcal{C}^2) - e_c(\mathcal{C}) \\ &\leq 2f(\epsilon)\,\text{Min-}k\text{-Uncut}(\mathcal{C})) \\ &= f(\epsilon)\,[\text{Min-}k\text{-CC}(\mathcal{C}) - (e_c(\mathcal{C}) - W)] \\ &\leq 2f(\epsilon)\,\text{Min-}k\text{-CC}(\mathcal{C})\,.\end{aligned}$$

The last inequality above follows from this reasoning

$$\begin{aligned}W - e_c(\mathcal{X}) &= \text{Min-}k\text{-Uncut}(\mathcal{X}) + \text{Max-}k\text{-Cut}(\mathcal{X}) - e_c(\mathcal{X}) \\ &= \text{Min-}k\text{-Uncut}(\mathcal{X}) + e_c(\mathcal{X}) - \text{Max-}k\text{-Cut}(\mathcal{X}) \\ &\qquad - 2(e_c(\mathcal{X}) - \text{Max-}k\text{-Cut}(\mathcal{X})) \\ &\leq \text{Min-}k\text{-CC}(\mathcal{X})\,,\end{aligned}$$

which is a consequence of Max-k-Cut$(\mathcal{X}) = \sum_{e \in E_c(\mathcal{X})} \delta(e) \leq \sum_{e \in E_c(\mathcal{X})} 1 = e_c(\mathcal{X})$. □

Lemma 15 tells us that $\mathcal{C}^2$ is a $(1 + O(\epsilon))$-approximation to $\mathcal{C}$ in Min-k-CC cost. Of course, $\mathcal{C}^2$ is unknown; however, it is a candidate solution that Phase Three could find. If Phase Three did a perfect job, it would find at least a $(1 + O(\epsilon))$-approximation to $\mathcal{C}$.

We now need to show that the approximate solutions that Phase Three obtains

on groups are sufficient for our purposes. Recall that g is a partitioning of $[k]$ into γ groups, with $m_j = |g^{-1}(j)|$.

Lemma 16. *Suppose $\mathcal{X}$ and $\mathcal{Y}$ are g-equivalent clusterings. If for each j,*

$$\text{MIN-}m_j\text{-CC}(\mathcal{X}|_j) \leq (1 + f(\epsilon)) \,\text{MIN-}m_j\text{-CC}(\mathcal{Y}|_j)$$

Then, over all of V,

$$\text{MIN-}k\text{-CC}(\mathcal{X}) \leq (1 + f(\epsilon)) \,\text{MIN-}k\text{-CC}(\mathcal{Y})\,.$$

Proof. Note that

$$\text{MIN-}k\text{-CC}(\mathcal{X}) = \sum_{j\in[\gamma]} \text{MIN-}m_j\text{-CC}(\mathcal{X}|_j) + \sum_{\mathcal{X}_(u)\not\equiv\mathcal{X}(v)} (1 - \delta(u, v))\,,$$

and similarly for $\mathcal{Y}$. The g-equivalence of $\mathcal{X}$ and $\mathcal{Y}$ tells us that the rightmost summation is common to both clusterings, leading to the statement of the lemma. □

By construction, $\mathcal{D}^3$ is equivalent to $\mathcal{D}^2$, which is equivalent to $\mathcal{C}^2$ by Remark 10.

Corollary 8. *Suppose that $\mathcal{D}^3|_j$ is a $(1+O(\epsilon))$-approximation to the optimal solution to* MIN-m_j-CC *over G_j^2. Then $\mathcal{D}^3$ is a $(1 + O(\epsilon))$-approximation to* MIN-k-CC *on G.*

It remains to show that, indeed, $\mathcal{D}^3$ is a good solution over each group. For the group of small clusters, $j = 0$, we just recurse the MIN-k-CC algorithm; since $|G_0^2| \leq \epsilon|V|$, we can apply an inductive argument.

We will now show that the MAX-m_j-CC PTAS of Section 5.3 provides a good MIN-m_j-CC solution on a group of large clusters. We would like to apply Lemma 10 immediately, but we need to find a similar result for MAX-k-CC and MIN-k-CC.

Lemma 17. *Let $\mathcal{C}$ be a clustering of G. Let $M = \max_i |C_i|$. Suppose that, for all i, $|C_i| \geq f(\epsilon)M$, and that*

$$g(\epsilon)\,\text{MAX-}k\text{-CUT}(\mathcal{C}) < \text{MIN-}k\text{-UNCUT}(\mathcal{C})\,, \tag{5.10}$$

then

$$\frac{f^2(\epsilon)g(\epsilon)}{2}\textsc{Max}\text{-}k\text{-CC}(\mathcal{C}) < \textsc{Min}\text{-}k\text{-CC}(\mathcal{C})\,. \tag{5.11}$$

Proof. For convenience, let

$$\begin{aligned} X &= \textsc{Max}\text{-}k\text{-Cut}(\mathcal{C}) & W &= e_c(\mathcal{C}) - X \\ Y &= \textsc{Min}\text{-}k\text{-Uncut}(\mathcal{C}) & Z &= e_u(\mathcal{C}) - Y\,, \end{aligned}$$

all positive quantities. Also, let f stand for $f(\epsilon)$ and g for $g(\epsilon)$. Then

$$X + W = e_c(\mathcal{C}) = \sum_{i<j} |C_i||C_j| \geq \sum_{i<j} f^2M^2 \geq f^2\frac{M^2k}{2}$$

And

$$Y + Z = e_u(\mathcal{C}) = \sum_i \binom{|C_i|}{2} \leq \sum_i \binom{M}{2} = k\binom{M}{2} \leq \frac{M^2k}{2}$$

Combining these, we have,

$$X + W \geq f^2(Y + Z) \geq f^2Z \tag{5.12}$$

Consider the following:

$$\begin{aligned} & f^2g(X+Z) - 2(Y+W) \\ &= g(f^2Z) + f^2gX - 2(Y+W) \\ &\leq g(X+W) + f^2gX - (f^2+1)Y - 2W && \text{by (5.12), and } f \leq 1 \\ &= (f^2+1)(gX - Y) + (g-2)W < 0 && \text{as } gX < Y, \text{ and } g \leq 1, \end{aligned}$$

which completes the proof. □

Corollary 9. *Let $\mathcal{C}|_j$ be the optimal solution to* Max-m_j-CC *on G_j^2, where $j > 0$. Then a $(1-O(\epsilon))$-approximate solution to $\mathcal{C}_j$ on G_j^2 will be a $(1+O(\epsilon))$-approximate solution to* Min-m_j-CC* *on G_j^2*

Proof. Each large cluster C_i has $|C_i| \geq e^{j_0}M$. Applying Lemma 10 and Lemma 17, we see that

$$\textsc{Max}\text{-}m_j\text{-CC}(\mathcal{C}|_j) < \Omega(\frac{1}{\epsilon})\textsc{Min}\text{-}m_j\text{-CC}(C|_j)$$

The C_i are optimal (on groups) so the above equation describes the relation between optimal solutions. An application of Remark 7 completes the proof. □

Theorem 6. *The algorithm in Section 5.4 is a PTAS for the* Min-k-CC *problem.*

Proof. Combining Corollary 9, Corollary 8, and the PTAS of Section 5.3 for metric-Max-k-CC, we have the required bound on the quality of the solution of $\mathcal{D}^3$. The running time of the algorithm is polynomial, as it is the combination of PTASes, the guessing of $O(k)$ values, and the solution of a Bipartite Matching problem. □

Note that our algorithm will call the FK Max-k-Cut PTAS once for each guess—it makes a total of $O(n^{3k}k^2)$ guesses, and passes a ϵ value of $\epsilon' = \epsilon^{3j_0+3}/90k^{18}$ through to that algorithm.

5.5 Conclusion

This chapter has described a PTAS for each of the maximisation and minimisation versions of metric-k-Correlation Clustering, a generalisation of the k-Consensus Clustering problem. The minimisation PTAS is of particular interest as both the unrestricted Consensus Clustering problem is known to be APX-hard. Until now, the 0/1 k-Correlation Clustering problem had been the only minimisation problem in the Correlation Clustering family known to have a PTAS.

Chapter 6

Conclusions

In this thesis, we have studied a number of combinatorial optimisation problems that share the common theme of advice. These problems asked us to take a graph, representing local pair-wise information about how vertices should be organised, and construct a global organisation (either a clustering or a ranking of the data).

In Chapter 3 we described the concept of relaxation; taking such a combinatorial problem and considering a non-integral version. We highlighted two particular techniques—the general method of semi-definite programming and the clustering-specific approach called Spectral Clustering. We considered how to apply such techniques to the Affinity Clustering problem when advice was incorporated. We described a series of algorithms which attempt to respect the advice to varying degrees without compromising the quality or running-time of the original relaxations. We conducted a series of experiments to conclude that these algorithms can out-perform algorithms which ignore the advice, or treat it naively.

In Chapter 4 we considered both local-search algorithms, and algorithms inspired by local-search, for two problems, Min Feedback Arc Set and Min-2-CC. We discovered algorithms which were as good as, or better than, the state of the art for these problems, and were able to demonstrate that some algorithms were not good approximators. Significantly, we proved that the PASTA-toss algorithm is a 2-approximation for the Min-2-CC problem on complete graphs.

In Chapter 5 we provide a polynomial time approximation scheme for the k-Consensus Clustering problem, a specialisation of the metric k-Correlation

Clustering problem. This is an important development for the Consensus Clustering problem, which was recently shown to be APX-hard for un-constrained minimisation. In doing so, we developed new techniques for ensuring correct cluster sizes based on finding matchings, which could have application to a variety of other restricted clustering problems.

Reflections

This thesis has demonstrated the strength of three key approaches for solving optimisation problems that incorporate advice: relaxation, local-search and sampling. These three approaches span the full range of perspectives we have chosen to focus on in this thesis.

Relaxation, especially to a spectral problem, is a very time-efficient and practically effective method of solving complex combinatorial problems. Although we have seen that relaxations to semi-definite programs can be (and often are) used to provide algorithms with approximation guarantees, the spectral relaxation is preferred due to its real-world performance, not its theoretical properties. Certainly a spectral algorithm with a performance guarantee would be a major breakthrough.

Local-search is another approach which is used more traditionally by the practical camp; however, one success of this thesis has been to find a local-search algorithm which not only performs well in practice, but also has an interesting guarantee. We anticipate that the method of seeding local-search algorithms with well chosen starting solutions could provide the theoretical community with further approximation algorithms that perform well in practice.

The sampling approach certainly leads to PTASes which are relatively easy to analyse from a theoretical perspective. This is of great use to the theoretical community. However, as we saw in chapter 4, most (if not all) PTASes' running times are unhelpfully high. It would be very interesting to see algorithms which draw ideas from successful PTASes, yet are able to run in reasonable amounts of time. In the best case scenario, such an algorithm would still be a PTAS, and could potentially perform very well in practice. Unfortunately our first attempts to do so (in chapter 4) had little success—we hope that future researchers have better luck.

Bibliography

[1] Agarwal, A., Charikar, M., Makarychev, K., and Makarychev, Y. $O(\sqrt{\log n})$ approximation algorithms for min UnCut, min 2CNF deletion, and directed cut problems. *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing* (2005), 573–581.

[2] Ailon, N., Charikar, M., and Newman, A. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing* (2005), ACM New York, NY, USA, pp. 684–693.

[3] Ali, I., Cook, W., and Kress, M. On the Minimum Violations Ranking of a Tournament. *Management Science 32*, 6 (1986), 660–672.

[4] Alon, N. Ranking tournaments. *SIAM Journal on Discrete Mathematics 20*, 1 (2007), 137–142.

[5] Arora, S., Frieze, A., and Kaplan, H. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. *Mathematical Programming 92*, 1 (2002), 1–36.

[6] Arora, S., Karger, D., and Karpinski, M. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences 58*, 1 (1999), 193–210.

[7] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. Proof verification and the hardness of approximation problems. *Journal of the ACM 45*, 3 (1998), 501–555.

[8] Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V. Local Search Heuristics for k-Median and Facility Location Problems. *SIAM Journal on Computing 33*, 3 (2004), 544–562.

[9] Aslam, J., and Montague, M. Models for metasearch. *Proceedings of the Twenty-Fourth Annual international ACM SIGIR Conference on Research and Development in Information Retrieval* (2001), 276–284.

[10] Asuncion, A., and Newman, D. UCI machine learning repository, 2007.

[11] Bang-Jensen, J., and Thomassen, C. A Polynomial Algorithm for the 2-Path Problem for Semicomplete Digraphs. *SIAM Journal on Discrete Mathematics 5* (1992), 366–376.

[12] Bansal, N., Blum, A., and Chawla, S. Correlation Clustering. *Machine Learning 56*, 1 (2004), 89–113.

[13] Bar-Hillel, A., Hertz, T., Shental, N., and Weinshall, D. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research 6*, 1 (2006), 937–965.

[14] Bar-Noy, A., and Naor, J. Sorting, Minimal Feedback Sets, and Hamilton Paths in Tournaments. *SIAM Journal on Discrete Mathematics 3* (1990), 7–20.

[15] Basu, S., Bilenko, M., and Mooney, R. A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), ACM New York, NY, USA, pp. 59–68.

[16] Bellman, R. Dynamic programming. *Science 153*, 3731 (1966), 34–37.

[17] Bertolacci, M., and Wirth, A. Are approximation algorithms for consensus clustering worthwhile? *Proceedings of the Seventh Annual International Conference on Data Mining* (2007), 437–442.

[18] Bonizzoni, P., Della Vedova, G., Dondi, R., and Jiang, T. On the approximation of correlation clustering and consensus clustering. *Journal of Computer and System Sciences 74*, 5 (2008), 671–696.

[19] Chanas, S., and Kobylański, P. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications 6*, 2 (1996), 191–205.

[20] Charbit, P., Thomassé, S., and Yeo, A. The Minimum Feedback Arc Set Problem is NP-Hard for Tournaments. *Combinatorics, Probability and Computing 16*, 01 (2006), 1–4.

[21] Charikar, M., Guruswami, V., and Wirth, A. Clustering with qualitative information. *Journal of Computer and System Sciences 71*, 3 (2005), 360–383.

[22] Charon, I., and Hudry, O. A survey on the linear ordering problem for weighted or unweighted tournaments. *4OR: A Quarterly Journal of Operations Research 5*, 1 (2007), 5–60.

[23] Chung, F. *Spectral Graph Theory*. American Mathematical Society, 1997.

[24] Conitzer, V., and Sandholm, T. Common voting rules as maximum likelihood estimators. In *Proceedings of the Twenty-First Annual Conference on Uncertainty in Artificial Intelligence* (2005), pp. 145–152.

[25] Cook, W., Golan, I., and Kress, M. Heuristics for ranking players in a round robin tournament. *Computers and Operations Research 15*, 2 (1988), 135–144.

[26] Coppersmith, D., Fleischer, L., and Rudra, A. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), ACM New York, NY, USA, pp. 776–782.

[27] Dantzig, G. *Linear Programming and Extensions*. Princeton Univ Pr, 1963.

[28] DasGupta, B., Enciso, G., Sontag, E., and Zhang, Y. Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. *BioSystems 90*, 1 (2007), 161–178.

[29] Davidson, I., and Ravit, S. Clustering With Constraints: Feasibility Issues and the fc-Means Algorithm. In *Proceedings of the Fifth International Conference on Data Mining* (2005), Society for Industrial Mathematics.

[30] De Bie, T. Deploying SDP for machine learning. In *Proceedings of the Fifteenth European Symposium on Artificial Neural Networks* (2007), pp. 205–210.

[31] De Bie, T., and Cristianini, N. Fast SDP Relaxations of Graph Cut Clustering, Transduction, and Other Combinatorial Problems. *The Journal of Machine Learning Research 7* (2006), 1409–1436.

[32] De Bie, T., Suykens, J., and De Moor, B. Learning from general label constraints. *Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition* (2004), 671–679.

[33] Demaine, E., Emanuel, D., Fiat, A., and Immorlica, N. Correlation clustering in general weighted graphs. *Theoretical Computer Science 361*, 2-3 (2006), 172–187.

[34] Demetrescu, C., and Finocchi, I. Removing cycles for minimizing crossings. *ACM Journal on Experimental Algorithmics (JEA) 6*, 1 (2001).

[35] Diestel, R. *Graph theory*, electronic ed. Springer-Verlag Heidelberg, New York, 2005.

[36] Dom, M., Guo, J., Huffner, F., Niedermeier, R., and Truss, A. Fixed-parameter tractability results for feedback set problems in tournaments. *Lecture Notes in Computer Science 3998* (2006), 320–331.

[37] Downey, R., and Fellows, M. *Parameterized Complexity*. Springer New York, 1999.

[38] Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. Rank aggregation methods for the web. In *Proceedings of the Tenth International Conference on the World Wide Web* (2001), ACM New York, NY, USA, pp. 613–622.

[39] Eades, P., Lin, X., and Smyth, W. A Fast and Effective Heuristic for the Feedback Arc Set Problem. *Information Processing Letters 47*, 6 (1993), 319–323.

[40] Eades, P., and Wormald, N. Edge crossings in drawings of bipartite graphs. *Algorithmica 11*, 4 (1994), 379–403.

[41] Even, G. Approximating Minimum Feedback Sets and Multicuts in Directed Graphs. *Algorithmica 20*, 2 (1998), 151–174.

[42] Fernandez de la Vega, W. MAX-CUT has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms 8*, 3 (1996).

[43] Fernandez de la Vega, W., Karpinski, M., Kenyon, C., and Rabani, Y. Approximation schemes for clustering problems. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (2003), pp. 50–58.

[44] Fernandez de la Vega, W., and Kenyon, C. A randomized approximation scheme for metric max-cut. *Journal of Computer and System Sciences 63*, 4 (2001), 531–541.

[45] Fiedler, M. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal 23*, 98 (1973), 298–305.

[46] Filkov, V., and Skiena, S. Integrating microarray data by consensus clustering. In *Proceedings of the Fifteenth IEEE International Conference on Tools with Artificial Intelligence* (2003), pp. 418–426.

[47] Fisher, D. Knowledge acquisition via incremental conceptual clustering. *Machine Learning 2*, 2 (1987), 139–172.

[48] Ford, L., and Fulkerson, D. Flows in Networks. *Princeton, New Jersey* (1962).

[49] Frieze, A., and Kannan, R. The regularity lemma and approximation schemes for dense problems. *Proceedings of the Thirty-Seventh Annual IEEE Symposium on Foundations of Computer Science* (1996), 12–20.

[50] Frieze, A., and Kannan, R. Quick Approximation to Matrices and Applications. *Combinatorica 19*, 2 (1999), 175–220.

[51] Garey, M., Graham, R., and Ullman, J. An analysis of some packing algorithms. In *Combinatorial algorithms (Courant Computer Science Symposium 9)* (1972), pp. 39–47.

[52] Gionis, A., Mannila, H., and Tsaparas, P. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data 1*, 1 (2007), 1–30.

[53] Giotis, I., and Guruswami, V. Correlation clustering with a fixed number of clusters. *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2006), 1167–1176.

[54] Goemans, M., and Williamson, D. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM 42*, 6 (1995), 1115–1145.

[55] Grotschel, M., Lovasz, L., and Schrijver, A. *Geometric Algorithms and Combinatorial Optimization, Algorithms and Combinatorics*. Springer, Berlin, 1988.

[56] Guo, J., Hüffner, F., and Moser, H. Feedback arc set in bipartite tournaments is NP-complete. *Information Processing Letters 102*, 2-3 (2007), 62–65.

[57] Hagen, L., and Kahng, A. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 11*, 9 (1992), 1074–1085.

[58] Harary, F. On the notion of balance of a signed graph. *Michigan Mathematical Journal 2* (1953), 143–146.

[59] Huffner, F., Betzler, N., and Niedermeier, R. Optimal edge deletions for signed graph balancing. *Proceedings of the Sixth Workshop on Experimental Algorithms* (2007), 297–310.

[60] Indyk, P. A sublinear time approximation scheme for clustering in metric-spaces. *Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science* (1999), 154–159.

[61] Johnson, D. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing* (1973), ACM New York, NY, USA, pp. 38–49.

[62] Johnson, D. Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal of Computing 4*, 1 (1975), 77–84.

[63] Kamvar, S., Klein, D., and Manning, C. Spectral learning. In *International Joint Conference On Artificial Intelligence* (2003), vol. 18, pp. 561–566.

[64] Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., and Wu, A. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications 28*, 2-3 (2004), 89–112.

[65] Karmarkar, N. A new polynomial-time algorithm for linear programming. *Combinatorica 4*, 4 (1984), 373–395.

[66] Karp, R. M. Reducibility among combinatorial problems. *Complexity of Computer Computations* (1972), 85–103.

[67] Kemeny, J. Mathematics without numbers. *Daedalus 88* (1959), 571–591.

[68] Kendall, M. Further Contributions to the Theory of Paired Comparisons. *Biometrics 11*, 1 (1955), 43–62.

[69] Kenyon-Mathieu, C., and Schudy, W. How to rank with few errors. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing* (2007), ACM New York, NY, USA, pp. 95–103.

[70] Khachiyan, L. A polynomial time algorithm for linear programming. In *Soviet Mathematics Doklady* (1979), vol. 20, pp. 191–194.

[71] Khot, S. On the power of unique 2-prover 1-round games. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing* (2002), ACM New York, NY, USA, pp. 767–775.

[72] Khot, S., Kindler, G., Mossel, E., and O'Donnell, R. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? In *Proceedings of the Forty-Fifth Annual IEEE Symposium on Foundations of Computer Science* (2004), pp. 146–154.

[73] Klein, D., and Randić, M. Innate degree of freedom of a graph. *Journal of Computational Chemistry 8*, 4 (1987), 516–521.

[74] Kumar, A., Sabharwal, Y., and Sen, S. A simple linear time (1+/spl epsiv/)-approximation algorithm for k-means clustering in any dimensions. In *Proceedings of the Forty-Fifth Annual IEEE Symposium on the Foundations of Computer Science* (2004), pp. 454–462.

[75] Land, A., and Doig, A. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society* (1960), 497–520.

[76] Liu, Y., Jin, R., and Jain, A. BoostCluster: boosting clustering by pairwise constraints. In *Proceedings of the Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2007), ACM Press New York, NY, USA, pp. 450–459.

[77] Lloyd, S. Least squares quantization in PCM. *Information Theory, IEEE Transactions on 28*, 2 (1982), 129–137.

[78] Mathieu, C., and Schudy, W. Yet another algorithm for dense max cut: Go greedy. *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2008), 176–182.

[79] McJones, P. Eachmovie collaborative filtering data set. *DEC Systems Research Center 249* (1997).

[80] Meila, M., Phadnis, K., Patterson, A., and Bilmes, J. Consensus ranking under the exponential model. In *Proceedings of the Twenty-Third Annual Conference on Uncertainty in Artificial Intelligence (to appear)* (2007).

[81] Meila, M., and Shi, J. A random walks view of spectral segmentation. *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics* (2001).

[82] Mezard, M., Parisi, G., and Angel, M. *Spin glass theory and beyond*. World Scientific.

[83] Mirkin, B. G., and Chernyi, L. B. On Measurement of Distance Between Partitions of a Finite Set of Units. *Automation and Remote Control 31*, 786–792.

[84] Nadler, B., and Galun, M. Fundamental limitations of spectral clustering. In *Advances in Neural Information Processing Systems* (Cambridge, MA, 2007), B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19, MIT Press, pp. 1017–1024.

[85] Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems 13* (2001), 849–856.

[86] Ostrovsky, R., Rabani, Y., Schulman, L., and Swamy, C. The effectiveness of lloyd-type methods for the k-means problem. In *Proceedings of the Forty-Seventh Annual IEEE Symposium on Foundations of Computer Science* (2006), IEEE Computer Society Washington, DC, USA, pp. 165–176.

[87] Paccanaro, A., Casbon, J., and Saqi, M. Spectral clustering of protein sequences. *Nucleic Acids Research 34*, 5 (2006), 1571–1580.

[88] Pachter, L., and Kim, P. Forcing matchings on square grids. *Discrete Mathematics 190*, 1-3 (1998), 287–294.

[89] Park, L., and Ramamohanarao, K. Mining web multi-resolution community-based popularity for information retrieval. In *Proceedings of the*

Sixteenth ACM Conference on Conference on Information and Knowledge Management (2007), ACM New York, NY, USA, pp. 545–554.

[90] Pelleg, D., and Baras, D. K-means with Large and Noisy Constraint Sets. *Lecture Notes in Computer Science 4701* (2007), 674–682.

[91] Poljak, S., and Tuza, Z. The max-cut problem: a survey. *Special Year on Combinatorial Optimization, DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society* (1995).

[92] Raghavendra, P. Optimal algorithms and inapproximability results for every CSP? In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing* (2008), ACM New York, NY, USA, pp. 245–254.

[93] Ritz, W. On a new method for solving some variational problems in mathematical physics. *Journal for Pure and Applied Mathematics 135* (1908), 1–61.

[94] Saab, Y. A Fast and Effective Algorithm for the Feedback Arc Set Problem. *Journal of Heuristics 7*, 3 (2001), 235–250.

[95] Shamir, R., Sharan, R., and Tsur, D. Cluster graph modification problems. *Discrete Applied Mathematics 144*, 1-2 (2004), 173–182.

[96] Shi, J., and Malik, J. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*, 8 (2000), 888–905.

[97] Solé, P., and Zaslavsky, T. A coding approach to signed graphs. *SIAM Journal on Discrete Mathematics 7* (1994), 544–553.

[98] Spielman, D., and Teng, S. Spectral partitioning works: Planar graphs and finite element meshes. *Linear Algebra and Its Applications 421*, 2-3 (2007), 284–305.

[99] Swamy, C. Correlation clustering: maximizing agreements via semidefinite programming. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2004), pp. 526–527.

[100] Szemeredi, E. Regular partitions of graphs. In *Problemes Combinatoires et Theorie des Graphes* (1978), pp. 399–401.

[101] Tarjan, R. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing 1* (1972), 114–121.

[102] Trinajstic, N. *Chemical graph theory.* CRC Press.

[103] van Zuylen, A., Hegde, R., Jain, K., and Williamson, D. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), pp. 405–414.

[104] Vazirani, V. *Approximation Algorithms.* Springer, 2001.

[105] Wagstaff, K., and Cardie, C. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning* (2000), pp. 1097–1097.

[106] Wilcoxon, F. Comparisons by ranking methods. *Biometric Bulletin 1* (1945), 80–82.

[107] Xing, E., and Jordan, M. *On Semidefinite Relaxation for Normalized K-cut and Connections to Spectral Clustering.* Computer Science Division, University of California, 2003.

[108] Xing, E., and Karp, R. CLIFF: clustering of high-dimensional microarray data via iterative feature filtering using normalized cuts. *Bioinformatics 17*, 90001 (2001), 306–315.

[109] Xing, E., Ng, A., Jordan, M., and Russell, S. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems* (2003), vol. 15, pp. 505–512.

[110] Yu, S., and Shi, J. Grouping with bias. In *Advances in Neural Information Processing Systems* (2001), vol. 13.

[111] Zaslavsky, T. Signed Graphs. *Discrete Applied Mathematics 4* (1982), 47–74.