Algorithms for Spatial Variations of Classic Optimization Problems with Healthcare Applications

by

Clarence Worrell

B.S. in Fire Protection Engineering, University of Maryland, 2001

M.S. in Fire Protection Engineering, University of Maryland, 2002

M.S. in Industrial Engineering, University of Pittsburgh, 2017

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2023

Algorithms for Spatial Variations of Classic Optimization Problems with Healthcare Applications

Clarence Worrell, PhD

University of Pittsburgh, 2023

Operating healthcare systems efficiently is critical for maximizing healthcare delivery to patients. Due to the inherent spatial structure in many healthcare systems, however, this requires solving challenging *spatial* optimization problems. In this work, we design solution algorithms that are implementation-friendly, scalable, and have theoretical performance guarantees for spatial variations of three classic optimization problems that arise in this context: machine assignment, facility location, and graph partitioning.

In Chapter 2, we consider minimizing makespan across parallel *machines that are located in different areas*, a problem motivated by multi-hospital systems that share laboratory workload using courier services. This system introduces opportunity to reduce makespan by assigning jobs to lesser-loaded, but spatially distant, machines at the expense of transportation time penalties. We develop an implementation-friendly list-scheduling algorithm and show that it is a 2-approximation algorithm when the machines are identical. Computational experiments show that the algorithm often performs much better than its worst-case guarantee, even when the machines are not identical.

In Chapter 3, we consider a variation of the k-center facility location problem in which facility agents can meet clients at intermediate locations (i.e., "*meet-in-the-middle*") to deliver goods. This problem arises when designing vaccination campaigns for geographically dispersed populations with limited transportation resources. We show that the meet-in-the-middle k-center problem reduces to the standard k-center problem under a certain notion of "distance." We develop an approximation algorithm and a mixed integer linear program (MILP) that both leverage this reduction. Computational experiments show that we can solve the MILP 90% faster than a MILP formulation that does not leverage the reduction.

In Chapter 4, we consider the problem of recovering the ground-truth communities of a vertex-attributed graph that are *locally distinguishable* in the sense that they exhibit stronger community structure in more local parts of the graph. The partitioning problem arises when labeling training data that contain responses generated from different latent sources, a major barrier to using machine learning in healthcare. We develop a community detection algorithm and establish conditions under which it *almost exactly* recovers ground-truth communities. Computational experiments demonstrate the advantages of our method over a benchmark spectral method.

Table of Contents

List of Tables

List of Figures

Preface

To those who have supported and encouraged me, especially my wife, Stephanie.

1.0 Introduction

Healthcare systems have inherent spatial structure. Services must be located such that patients and providers can feasibly connect. Provider conglomeration introduces spatial structure through economies of scale, such as resource and workload sharing across distant provider locations. Underlying spatial attributes can even influence the structure of social networks used for provider advertising and public health messaging. In each of these contexts, it is important to be efficient in order to maximize utility, which is ultimately the providing of healthcare services and information to patients.

Motivated by problems that arise in healthcare practice, we study spatial variations of three classic optimization problems. First, we examine the *parallel machine assignment problem* in which the machines are located in different areas. Next, we study an extension of the *facility location problem* in which clients can meet facility agents at intermediate locations to receive goods and services. Finally, we consider the *graph partitioning problem* for graphs in which distinguishability between communities is stronger in local areas than it is globally.

These optimization problems are NP-hard, and problem instances observed in practice can be difficult to solve, even with modern computing hardware and commercial solvers. In our research, we seek solution algorithms that have following traits:

- **Implementation-friendly.** The algorithm should ideally be accessible to those without optimization expertise. When this is not possible, the algorithm should ideally not require problem reformulation, for example to implement a decomposition or relaxation strategy. Finally, the individual algorithm steps should ideally use well-established methods with vetted open-source implementations.
- **Scalable.** The algorithm should enable solving, either exactly or approximately, larger instances than could otherwise be achieved by solving the problem directly.
- **Performance-guaranteed.** The algorithm should ideally be accompanied by a proven worst-case performance guarantee.

We summarize our main contributions in each of the following sections.

1.1 Minimizing Makespan across Preloaded Machines

We consider the problem of minimizing makespan across parallel machines that are located in different areas. Each machine has been preloaded with a set of jobs that are located at the machine. The system manager has a one-time opportunity, prior to the start of processing, to reassign any of the preloaded jobs to any of the other machines. Reassigned jobs are transported *en masse* in a fixed amount of time and must be returned to their original machine after they are processed. Job processing times are deterministic, known prior to reassignment, and may be machine-specific.

The manager seeks a reassignment plan that minimizes makespan (i.e., the time at which all job processing has completed, and any reassigned jobs have been returned to their original machines). To support this managerial goal, we make the following contributions:

- We develop a MILP to identify optimal reassignment policies.
- We present a two-stage list scheduling algorithm and show it yields a 2-approximation guarantee when the machines are identical. We also show that a modified version of the algorithm admits an improved approximation guarantee when there is a large number of small jobs.
- We present computational results demonstrating that the two-stage list scheduling algorithm often performs much better than its worst-case performance guarantee, even when the machines are not identical. We also explore the efficient frontier between makespan and reassignment volume, and investigate the extent to which the MILP can be solved using a commercial solver.

As a motivating healthcare application, consider a multi-hospital system that shares clinical laboratory functions across the *spatially-distant hospitals* using a courier service. Jobs originate at each hospital in the morning, and the manager has a one-time opportunity to reassign them. Reassigned jobs are transported to their assigned hospitals for processing, and this transportation of course requires time. The manager would like to reassign jobs in a manner that minimizes the time at which all job processing completes. Implementation-friendliness is particularly important in this application because the manager is unlikely to have access to personnel with optimization expertise.

1.2 The Meet-in-the-Middle k-Center Problem

We study an extension of the classic facility location problem in which facilities can send agents to intermediate locations to *meet-in-the-middle* with clients. We consider different cost regimes by introducing a new notion of "distance" between locations that depends on the receiving and retrieving costs. Our notion of distance is a general notion, and it does not necessarily define a metric between locations (i.e., by satisfying both symmetry conditions and the triangle inequality), even when the receiving and retrieving costs individually define metrics. We develop a polynomial-time algorithm for the subproblem of optimally determining which client locations should receive and retrieve goods given a set of facility locations. From these ideas, we show how to reduce (in polynomial time) the meet-in-the-middle k-center problem to a (not necessarily metric) k-center problem.

Consider a system designer who seeks to minimize cost by identifying the set of locations in which to open facilities; the set of locations that should retrieve goods from facility locations; the set of locations that should receive goods from facility agents, and the set of locations that should retrieve goods from locations that receive goods from facility agents (i.e., meet-in-the-middle). To support this design goal, we make the following contributions:

- We present two MILP formulations for the meet-in-the-middle k-center problem; one of the formulations utilizes the reduction.
- We present an approximation algorithm for the meet-in-the-middle k-center problem that extends a greedy approximation algorithm for the metric k-center problem.
- Through computational experiments, we evaluate the performance of our methods and explore the empirical benefits of accounting for the ability to meet-in-the-middle in facility location.

As a motivating healthcare application, consider public health officials who are designing a vaccination campaign for a low income country in which the patient populations are *geographically dispersed*. Rather than solely installing fixed vaccination clinics, they would like to extend coverage by using an outreach strategy. During outreach, healthcare workers take vaccines from clinics to other intermediate locations, and patients travel to those intermediate locations to become vaccinated (i.e., the healthcare workers and patients "meet in the middle"). The campaign planners therefore need to determine where to locate healthcare facilities; identify which locations should receive outreach services; and amongst the locations that do not receive outreach services, identify which locations their patients should travel to for vaccination. Scalability is particularly important to facilitate designing country-scale vaccination campaigns.

1.3 Recovering Locally Distinguishable Communities with Applications to Clustering Training Data

In Chapter 4, we consider the problem of recovering the ground-truth communities of a vertex-attributed graph that are *locally distinguishable* in the sense that the ground-truth communities exhibit stronger community structure in more local parts of the graph (i.e., in subgraphs induced by vertices whose attributes are closer). As we demonstrate, the community detection problem arises when labeling training data that contain responses generated from different latent sources, a major barrier to using machine learning methods in practice, as it is costly to have subject matter experts label the data.

Locally distinguishable communities, however, may not be globally distinguishable, posing a challenge for standard community detection methods. To address this challenge, we make the following contributions :

- We develop a community detection algorithm that performs well on graphs exhibiting the local distinguishability property, compared to a standard spectral partitioning method.
- We develop the local stochastic block model (LSBM), which is a generalization of the stochastic block model (SBM) that generates graph instances exhibiting the local distin-

guishability property.

- We establish conditions under which our algorithm *almost exactly* recovers ground-truth communities in LSBM instances.
- Finally, our computational experiments demonstrate the advantages of using the method to cluster training data over a benchmark spectral method.

As a motivating healthcare application, consider using classification models to predict whether patients are adhering to their medication prescriptions [95]. Such models could afford healthcare providers an opportunity to mitigate the poor adherence rates observed in practice (for example, one meta-analysis [58] observed a 57% adherence rate to cardiovascular disease prevention drugs across 376,162 patients). However, the *labeled data* (i.e., data that identifies the ground truth of whether patients are adherent) necessary to construct such models is often not readily available or even obtainable [29, 43]. Our community detection algorithm provides a method to estimate ground-truth labels, which can then be used to construct medication adherence classification models. Furthermore, because our algorithm is graph-based, it is able to naturally leverage (through transductive learning) any partially labeled data that are available (for example a subset of data that are labeled by subject matter experts).

1.4 Organization

Chapters 2, 3, and 4 present the details of each contribution, including definition of the novel problem, algorithm presentation and analysis, and computational experiments. Chapter 5 concludes the dissertation by providing some areas for future research, as well a summary of how each algorithmic contribution achieves our intended traits: implementation-friendly, scalable, and performance-guaranteed.

2.0 Minimizing Makespan across Pre-Loaded Machines

2.1 Introduction

2.1.1 Problem Description

Consider m parallel machines in different locations, each of which is preloaded with a set of jobs that may be processed in any order. More specifically, let there be n jobs in total with jobs $J_i \subset [n] := \{1, \ldots, n\}$ preloaded on machine $i \in [m] := \{1, \ldots, m\}$. The collection of job subsets $J_1, \ldots, J_m$ partitions $[n]$, and we let $n_i := |J_i|$ denote the number of jobs preloaded on machine i. Processing job $j \in [n]$ on machine $i \in [m]$ requires $p_{ij} \in \mathbb{N}$ units of processing time. Note that the processing time of a job is independent of the machine on which it is preloaded. Before job processing starts, a decision-maker has a one-time opportunity to reassign any jobs to any other machines. Processing times are deterministic, known prior to the reassignment decision, and can be machine-specific when the machines are unrelated (i.e., not identical). The processing of retained preloaded (i.e., "domestic") jobs starts immediately at time $t = 0$.

Jobs reassigned from one machine to another are transported at time $t = 0$ and arrive at their receiving machine τ time units later, at which point the receiving machine can preempt the processing of its current job (if the machine is busy) and begin processing the "foreign" jobs it receives. When a machine finishes processing its foreign jobs, they are then returned to their originating machine, where they arrive τ time units later. The goal is to find a reassignment policy that minimizes makespan, i.e., the time until all jobs have been processed and all reassigned jobs have been returned to their originating machine. Let C^*_{max} denote the optimal makespan.

Note that, in order to minimize makespan, it is always optimal for a receiving machine to preempt the processing of its current domestic job at time $t = \tau$ (if the machine is busy); then process all of its foreign jobs (in any order); and finally complete processing the balance of its domestic jobs (also in any order). Second, while we find it convenient for clarity of exposition to assume that all foreign jobs are returned to their origins after the machine to which they are assigned completes their processing, returning the foreign jobs individually as they are completed (given enough transportation resources to do so) yields an equivalent makespan because the transportation time τ is constant.

Figure 1 (a) depicts a problem instance with two identical machines, an initial workload of seven jobs at machine 1 (dashed lines), and an initial workload of five jobs at machine 2 (solid lines). With no reassignments, the completion times of machine 1 and 2 are 12 and 16, respectively, resulting in a makespan of $\max\{12, 16\} = 16$. Figure 1 (b) depicts the policy that reassigns job 1 of duration two from machine 1 to machine 2 and job 10 of duration four from machine 2 to machine 1. The assumed transportation time is $\tau = 4$, and the grey shading indicates periods when jobs are being transported. Machine 1 preempts job 4 upon receiving foreign job 10, and it finishes job 4 after completing foreign job 10. This policy generates an optimal makespan of $C^*_{max} = 14$. (Note that there are multiple polices that could generate this optimal makespan.) Figure 1 (c) depicts a sub-optimal policy that reassigns four jobs (1, 4, 5, 7) from machine 1 to machine 2 and three jobs (10, 11, 12) from machine 2 to machine 1; under this policy, idling occurs while waiting for jobs to be returned to their originating machine.

As a motivating application, consider, for example, a multi-hospital system where clinical pathology functions are shared across the hospitals using a courier service. Each job is a human tissue specimen to be processed. Jobs vary in complexity, and processing times vary by hospital because each hospital has their own specialties. The jobs originate at the hospitals in the morning, and the manager has a one-time opportunity to reassign them. Reassigned jobs are transported to the other hospitals, which of course requires time. Naturally, the manager would like to reassign jobs in a manner that minimizes makespan.

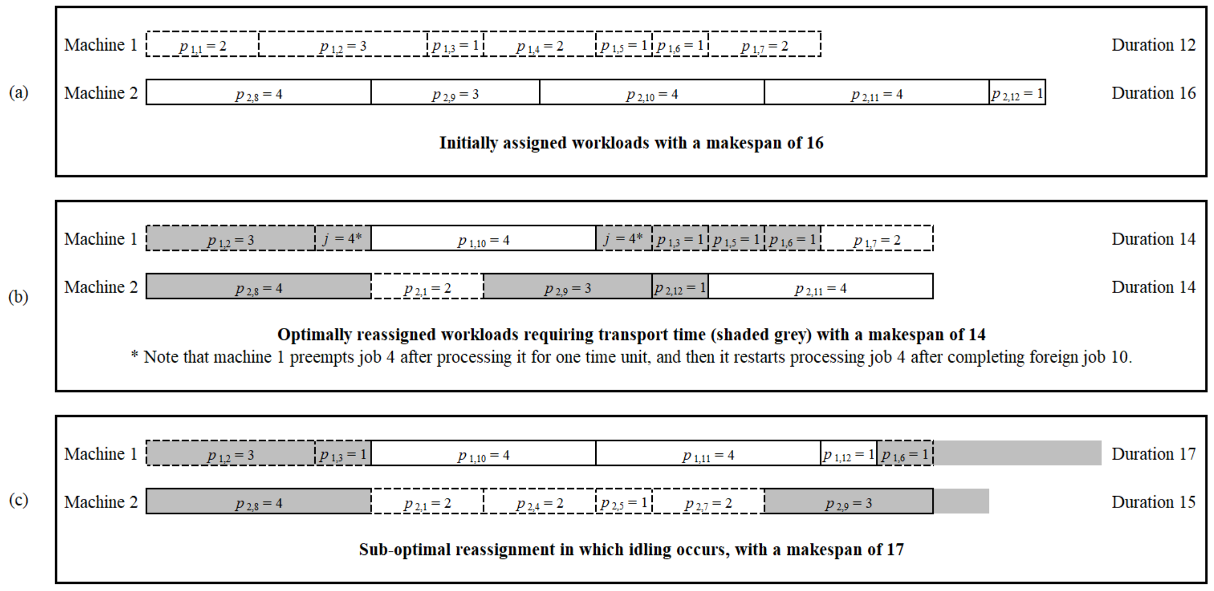


Figure 1: Example timelines for two identical machines and $\tau = 4$ depicting (a) initial workloads, (b) optimally reassigned workloads, and (c) a sub-optimal reassignment in which idling occurs.

The contributions of this work are as follows:

- In Section 2.2 we develop a mixed-integer linear program (MILP) to identify optimal reassignment policies.
- We present a *two-stage list scheduling algorithm* in Section 2.3 and show it yields a 2-approximation guarantee when the machines are identical. We also show that a modified version of the algorithm admits an improved approximation guarantee when there are a *large* number of *small* jobs.
- Finally, in Section 2.4 we present computational results that demonstrate that the two-stage list scheduling algorithm often performs much better than its worst-case performance guarantee, even when the machines are not identical. We also explore the efficient frontier between makespan and reassignment volume, and investigate the extent to which the MILP can be solved using a commercial solver.

2.1.2 Literature Review

2.1.2.1 Related Parallel Machine Scheduling Models

The problem of interest is a generalization of the classical parallel machine assignment problem, commonly denoted as $P||C_{max}$ when the machines are identical and $R||C_{max}$ when the machines are unrelated, using the notation of [36]. More specifically, when reassignment is instantaneous (i.e., $\tau = 0$), the problem of interest is exactly the classical parallel machine assignment problem. Many other variations of this problem have been considered in the literature; see, for example, the early works of [54], [35], [36], the more recent survey of [64], and the comprehensive handbook of [12]. Here we review the work that is most relevant to our problem.

Our problem can be interpreted as a variation of the machine assignment problem that involves jobs with *machine-specific ready and delivery times.* The ready time of a job is the time at which the job becomes available for processing. The delivery time of a job is the amount of time that it takes to send the job to its destination after it is processed by a machine; jobs are assumed to be delivered individually immediately after they are processed. Existing literature on ready and delivery times assumes that the machines are in the same location, and consequently assumes that each job's ready time and delivery time does not depend on the machine to which it is assigned. However, in our problem, the machines are in different locations, and the ready and delivery times can vary by machine. More specifically, a job is immediately ready to be processed by the machine at which it originates, and is ready to be processed on any other machine at time τ. Furthermore, the delivery time of a job is zero if the job is processed by the machine at which it originates, and τ otherwise.

[18] and [10] consider ready times together with due dates, [51] consider ready times together with setup times, [65] considers delivery times in the context of one machine, and [86] considers delivery times in the context of multiple machines. To the best of our knowledge, variations of the machine assignment problem with machine-specific ready and delivery times have yet to be considered in the literature. Furthermore, these ready times and delivery times have yet to simultaneously be considered together.

Returning processed foreign jobs to their originating machine shares some attributes with the integrated production and outbound distribution scheduling literature; see the survey [17]. This body of work involves a manufacturer with one or more plants that receive a set of orders from customers at various locations. Each order must be processed on one or more machines and then delivered individually or in batches to customers using one or more vehicles. This problem is in some ways more complex than ours and in other ways simpler. More specifically, it takes into account delivery vehicle routing, whereas we assume vehicles are immediately available to directly return processed jobs to their originating machine. On the other hand, this body of work does not consider the upfront transportation time present in our problem when jobs are reassigned.

Also related is the body of work that considers transportation time within flow shop environments; see, for example, [46], [47], and [94]. The basic problem involves a set of jobs that must be processed sequentially through a set of machines, each of which performs an operation in the overall processing of a given job. Partially completed jobs are moved between machines by interstage transporters in batches, and each shipment from one machine to another requires time. The objective is to determine the sequence in which jobs should be processed to minimize makespan. This problem is different from ours because it determines an *order* to process jobs through a *series* of machines, whereas our problem determines the *assignment* of jobs to *parallel* machines.

2.1.2.2 Related Scheduling Methodologies

[48] established that $P||C_{max}$ is NP-hard. Hence, the problem of minimizing makespan across preloaded machines and other generalizations of $P||C_{max}$ are NP-hard as well. Existing scheduling methodologies for these other generalizations are primarily based on integer programming formulations and approximation algorithms. *List scheduling algorithms* are an important class of these approximation algorithms that order the jobs into a list and then assign the jobs in that order to the machines. Each job is assigned to the machine that currently has the smallest workload. List scheduling algorithms typically do not achieve the best approximation factors, but they are easy for practitioners to implement.

Lastly, we summarize some of the methodological results most closely related to this work. List scheduling with an arbitrarily ordered list admits a 2-approximation guarantee for $P||C_{max}$; see [80] for a proof of this well-known result. [35] showed that list scheduling with an ordered list (from largest to smallest processing time) has an approximation factor of $\frac{4}{3} - \frac{1}{3m}$ for $P||C_{max}$. [86] developed a $(2 - \frac{2}{m+1})$-approximation algorithm for parallel machine scheduling on identical machines with delivery times (that do not vary by machine). To the best of our knowledge, approximation algorithms for parallel machine scheduling with ready times have yet to be developed; however, [51] present a MILP formulation for parallel machine scheduling with setup and ready times (that do not vary by machine).

2.2 MILP Formulation

We present an MILP formulation for the problem of minimizing makespan across preloaded machines. Recall that the decision to reassign jobs occurs at time $t = 0$, and all reassigned jobs arrive at their newly assigned machines at time $t = \tau$. Without loss of generality, we assume that any machine receiving foreign jobs immediately starts processing the foreign jobs, and that any domestic job in progress is preempted (stopped and restarted later without a time penalty). When a receiving machine completes its foreign jobs, they are returned to their originating machine(s), where they arrive after another τ time units. Finally, each machine completes any remaining domestic work.

First, let us discuss the decision variables of the MILP. For each $i, j \in [m] \times [n]$, we introduce a binary decision variable x_{ij} that indicates if job j is assigned to machine i for processing. Furthermore, for each $i \in [m]$, we introduce an auxiliary variable ξ_i that indicates if at least one job that is assigned to machine i is preloaded at a different machine.

Next, we derive an expression for makespan in terms of the decision variables x, ξ, and model parameters. To this end, we first define the *completion time* C_i of machine $i \in [m]$ to be the point in time at which (*i*) all of the jobs assigned to machine i have been processed and (*ii*) all of the foreign jobs assigned to machine i have been returned to their originating machine. It follows that the makespan of the system is $\max_{i \in [m]} C_i$. Notice that we can express the completion time of machine i as

$$C_i = \sum_{j \in [n] \setminus J_i} p_{ij} x_{ij} + \max \left\{ \sum_{j \in J_i} p_{ij} x_{ij},\ 2\tau\xi_i \right\}. \tag{2-1}$$

Thus, we have obtained an expression for the makespan in terms of x, ξ, and the model parameters.

We present the following MILP formulation (2-2) for the problem of minimizing makespan across preloaded machines.

$$\begin{aligned}
\min \quad & \mu \\
\text{s.t.} \quad & \mu \geq \sum_{j \in [n] \setminus J_i} p_{ij} x_{ij} + w_i && \forall i \in [m], && \text{(2-2a)} \\
& w_i \geq \sum_{j \in J_i} p_{ij} x_{ij} && \forall i \in [m], && \text{(2-2b)} \\
& w_i \geq 2\tau \xi_i && \forall i \in [m], && \text{(2-2c)} \\
& \xi_i \geq x_{ij} && \forall i \in [m],\ j \in [n] \setminus J_i, && \text{(2-2d)} \\
& \sum_{i=1}^{m} x_{ij} = 1 && \forall j \in [n], && \text{(2-2e)} \\
& x_{ij} \in \{0,1\} && \forall i,j \in [m] \times [n], && \text{(2-2f)} \\
& \xi_i \in \{0,1\} && \forall i \in [m], && \text{(2-2g)} \\
& \mu \in \mathbb{R}, && && \text{(2-2h)} \\
& w_i \in \mathbb{R} && \forall i \in [m], && \text{(2-2i)}
\end{aligned}$$

where μ and w are additional auxiliary variables. Constraints (2-2a)-(2-2c) model the makespan of the system using expression (2-1). Constraints (2-2d) guarantee that $\xi_i = 1$ when at least one job that is assigned to machine i is preloaded at a different machine. Constraints (2-2e) enforce that each job is processed by a machine.

We observe in our computational study presented in Section 2.4 that formulation (2-2) can unnecessarily reassign jobs when the same makespan can be achieved with fewer reassignments. Accordingly, we introduce the bi-criteria objective function

$$\mu + \epsilon \sum_{i \in [m]} \sum_{j \in [n] \setminus J_i} x_{ij}, \qquad \text{(2-3)}$$

where $\epsilon > 0$ is a *penalty parameter* that is used to penalize excessive job reassignment. Objective function (2-3) affords the decision-maker an opportunity to strike a balance between their desire to minimize makespan and their tolerance for job reassignment. If the objective is purely to minimize makespan, then choosing a small enough value of ϵ would ensure the MILP always returns the makespan-minimizing policy with the fewest number of reassignments. Our third computational study in Section 2.4 (see Figure 4) explores the efficient frontier between between makespan and the number of reassigned jobs, a practical consideration for the decision-maker.

2.3 Two-Stage List Scheduling Algorithm

In this section we present a two-stage list scheduling algorithm for the problem of minimizing makespan across preloaded machines; see the text following Algorithm 2-1 for a description of the algorithm.

Algorithm 2-1 Two-stage list scheduling algorithm for minimizing makespan across preloaded machines.

```
Input: Job subsets $J_i$, $i \in [m]$, processing times $p_{ij}$, $i, j \in [m] \times [n]$, and transportation
    time $\tau$.

1: Stage 1. For each machine $i \in [m]$:
     a. Order the jobs originating at machine $i$ into a list from largest to smallest processing
        time.
     b. Assign to machine $i$ the jobs from its list in order from largest to smallest until either
        $(i)$ all of the jobs originating at machine $i$ have been assigned to machine $i$, or $(ii)$
        the workload assigned to machine $i$ exceeds $2\tau$.
   Terminate the algorithm if Stage 1 assigns all jobs, else proceed to Stage 2.
2: Stage 2.
     a. Combine all unassigned jobs into an arbitrarily ordered list.
     b. Assign the jobs one-by-one from the list to the machine whose assignment increases
        the current makespan the least.
3: End
```

Stage 1 of Algorithm 2-1 constructs a job list for each machine $i \in [m]$ of the jobs J_i that are preloaded at machine i. The algorithm orders the jobs from largest to smallest processing time and then assigns them to machine i in this order until no jobs are left or the workload exceeds 2τ. The purpose of Stage 1 is to reduce the machine idle time that arises from waiting for reassigned jobs to arrive from other machines. In particular, if all machines are preloaded with at least 2τ time units of work (i.e., $\sum_{j\in J_i} p_{ij} \geq 2\tau$ for all $i \in [m]$), then Stage 1 ensures that no idle time occurs.

Stage 2 of Algorithm 2-1 constructs an arbitrarily ordered list of the jobs that are not assigned in Stage 1. The algorithm then assigns the jobs one-by-one from the list to the machine whose assignment increases the current makespan the least.

Further motivation for the two stages of Algorithm 2-1 is as follows. Consider applying an existing list scheduling algorithm, say the the algorithm of [35] that orders jobs from largest to smallest processing time, to the problem of interest. Generally speaking, when τ is large compared to the job processing times, Algorithm 2-1 avoids reassigning jobs, whereas Graham's algorithm would attempt to equivalently load each machine, potentially leading to large makespans; see Example 2.3.1.

Example 2.3.1. Consider a system with $m = 2$ machines, $n_1 = 1$ job preloaded at machine 1, $n_2 = 3$ jobs preloaded at machine 2, $p_{ij} = p$ for all i, j, and $\tau > p$. Algorithm 2-1 would assign each of the four jobs to the machine on which they are preloaded, generating a makespan of $3p$. Graham's algorithm, however, would assign one job preloaded on machine 2 to machine 1, and each of the remaining three jobs to the machine on which they are preloaded, generating a makespan of $2\tau + p > 3p$.

In Theorem 2.3.1 below we establish that Algorithm 2-1 achieves a 2-approximation guarantee when the machines are identical.

Theorem 2.3.1. *Algorithm 2-1 is a 2-approximation algorithm for the problem of minimizing makespan across preloaded identical machines.*

Proof. Let C_{max}^{Alg1} be the makespan of the assignment policy generated by Algorithm 2-1; l_i be the processing time of the last job assigned to machine i; and C_i^- be the completion time as defined by (2-1) for all jobs assigned to machine i by Algorithm 2-1, excluding the last job assigned to machine i.

First, suppose the algorithm terminates before implementing Stage 2. Without loss of generality, suppose machine k has the largest initial workload. If the algorithm terminates before Stage 2, all jobs are assigned to their originating machine. Hence,

$$C_{max}^{Alg1} = \max_{i\in[m]} \left\{C_i^- + l_i\right\} = C_k^- + l_k < 2\tau + l_k, \tag{2-4}$$

where the inequality follows from the algorithm terminating prior to reaching Stage 2. For the sake of contradiction, suppose $C_{max}^{Alg1} > C_{max}^*$, where C_{max}^* is the optimal makespan. Then, in the optimal assignment, a job preloaded at the initially most loaded machine k, say job j, must be assigned to a different machine, say machine i. It follows that

$$C_{max}^* \geq C_i^* \geq 2\tau + p_{ij} \geq 2\tau + l_k > C_{max}^{Alg1},$$

where C_i^* is the makespan of machine $i \in [m]$ under the optimal policy. The second inequality follows from the need to transport job j from machine k to i for processing and then return, the third inequality follows from the Stage 1 descending sort, and the final inequality follows from (2-4). We have obtained a contradiction.

Now suppose that the algorithm does not terminate before implementing Stage 2. Then we claim that

$$C_{max}^* > 2\tau. \tag{2-5}$$

Suppose not. Then, all jobs are assigned to their originating machine in the optimal assignment. However, in this setting, the algorithm would terminate before reaching Stage 2. Hence a contradiction and the claim holds.

Let us now consider a machine $i \in [m]$. We will show $C_{max}^{Alg1} \leq 2C_{max}^*$, implying Algorithm 2-1 is a 2-approximation algorithm. Suppose that $C_i^- \leq 2\tau$. Then

$$C_i^{Alg1} \leq 2\tau + l_i \leq 2C_{max}^*,$$

where C_i^{Alg1} denotes the completion time of machine i under the Algorithm 2-1 assignment policy, and the second inequality follows from (2-5) and the fact that $l_i \leq C_{max}^*$.

Now suppose that $C_i^- > 2\tau$, which implies the last job assigned to machine i is assigned in Stage 2. We will show that $C_i^- \leq C_{max}^*$, which will imply

$$C_i^{Alg1} = C_i^- + l_i \leq 2C_{max}^*.$$

For the sake of contradiction, suppose $C_i^- > C_{max}^*$. We claim $C_k^{Alg1} > C_{max}^*$ for all $k \in [m]$. We have two cases:

Case 1. If $k = i$, then $C_k^{Alg1} > C_k^- > C_{max}^*$.

Case 2. Suppose $k \neq i$. Then $C_k^{Alg1} > C_{max}^*$ because otherwise Stage 2 of Algorithm 2-1 would have instead assigned the last job that was assigned to machine i to machine k.

Let us now partition the machine indices $[m]$ into two sets:

$$M_1 := \left\{ k \in [m] : \sum_{j \in J_k} p_{kj} < 2\tau \right\} \text{ and } M_2 := [m] \setminus T_1.$$

Furthermore, for $k \in M_1$, define

$$\alpha_k := 2\tau - \sum_{j \in J_k} p_{kj},$$

which is the minimum amount of time that machine $k \in M_1$ can idle under any assignment. Note that jobs can be assigned so that machines indexed by M_2 do not idle. It follows that

$$\sum_{k=1}^{m} \sum_{j \in J_k} p_{kj} \leq mC_{max}^* - \sum_{k \in M_1} \alpha_k. \tag{2-6}$$

In the assignment policy generated by Algorithm 2-1, it follows from Stage 1 that each machine $k \in M_1$ runs for $C_k^{Alg1} - \alpha_i$ time units, and each machine $k \in M_2$ does not idle. Hence,

$$\sum_{k=1}^{m} \sum_{j \in J_k} p_{kj} = \sum_{k=1}^{m} C_k^{Alg1} - \sum_{k \in M_1} \alpha_k > mC_{max}^* - \sum_{k \in M_1} \alpha_k, \tag{2-7}$$

where the strict inequality follows from $C_k^{Alg1} > C_{max}^*$ for all $k \in [m]$. Inequalities (2-6) and (2-7) yield a contradiction. □

Stage 2 of Algorithm 2-1 utilizes an arbitrarily ordered list; however, existing literature on list scheduling establishes that there are practical and theoretical benefits to using ordered lists (see Subsection 2.1.2.2 for further discussion). While it is not immediately clear how to modify the proof of Theorem 2.3.1 to leverage such additional structure in general, it is not hard to see how to use ordered lists to obtain an improved performance guarantee when there are a large number of small jobs; see Theorem 2.3.2.

Theorem 2.3.2. *Suppose that $\tau \geq p_{ij}$ for all $i, j \in [m] \times [n]$ and $\tau \leq \alpha \frac{1}{m} \sum_j p_{ij} \leq \alpha C^*_{max}$ for some $\alpha > 0$. Also suppose that the machines are identical and Algorithm 2-1 constructs an ordered list (from largest to smallest processing time) in Step 'a' of Stage 2 instead of an arbitrarily ordered list. Then this modified version of Algorithm 2-1 achieves a factor* $\min\{2, 3\alpha + \frac{4}{3} - \frac{1}{3m}\}$ *approximation guarantee.*

Proof. By Theorem 2.3.1, it is sufficient to show that Algorithm 2-1 is a $(3\alpha + \frac{4}{3} - \frac{1}{3m})$-approximation algorithm. Also, if Algorithm 2-1 terminates before Stage 2, then the same argument used in the proof of Theorem 2.3.1 shows that Algorithm 2-1 constructs an optimal schedule; accordingly, we assume that Algorithm 2-1 does not terminate before Stage 2.

The jobs assigned during Stage 1 are processed within 3τ time units on each machine due to the assumption that $\tau \geq p_{ij}$ for all $i, j \in [m] \times [n]$. Consider the following schedule that has a larger makespan than the schedule that Algorithm 2-1 constructs. Suppose that each machine $i \in [m]$ idles after processing its Stage 1 jobs until time $t = 3\tau$. Then the problem of assigning the remaining Stage 2 jobs (which will start being processed at time $t = 3\tau$) is an instance of $P||C_{max}$. Let $C^{*,2}_{max}$ denote the optimal makespan of this instance. Suppose that we use the list scheduling algorithm of [35] (i.e., order the jobs from largest to smallest processing time) to approximately solve this assignment problem. It is not hard to see that the makespan of this schedule is greater than the makespan of the schedule that Algorithm 2-1 constructs.

Thus, because Graham's algorithm is a $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithm for $P||C_{max}$, the makespan of the schedule that Algorithm 2-1 constructs is no greater than

$$3\tau + \left(\frac{4}{3} - \frac{1}{3m}\right) C^{*,2}_{max} \leq 3\tau + \left(\frac{4}{3} - \frac{1}{3m}\right) C^*_{max} \leq \left(3\alpha + \frac{4}{3} - \frac{1}{3m}\right) C^*_{max},$$

where the first inequality follows from the fact that $C^{*,2}_{max} \leq C^*_{max}$, and the second inequality from the condition that $\tau \leq \alpha C^*_{max}$. We have established the desired result. □

Under the conditions of Theorem 2.3.2, we see that using an ordered list in Step 'a' of Stage 2 leads to an improved approximation guarantee when the transportation time contributes to less than $\alpha = \frac{2}{9} + \frac{1}{9m} > \frac{2}{9}$ of the optimal makespan. We also see that the approximation guarantee improves with decreasing number of machines m and with decreasing α (i.e., when transportation time is small relative to the optimal makespan).

In practical applications, the decision-maker may prefer retaining some quantity of jobs at the originating machines, for example due to economic or other costs not explicitly modeled. Modifying Algorithm 2-1 to allow a choice of when to terminate Stage 1, would give the decision-maker the opportunity to strike a balance between their tolerance for reassigning jobs and the performance guarantee for makespan minimization.

While Theorem 2.3.1 and Theorem 2.3.2 present bounds on the worst-case performance, in practical applications the performance can be much better than the worst-case bounds, as illustrated in Section 2.4.

2.4 Computational Experiments

In this section we study the empirical approximation ratio of Algorithm 2-1, the performance of Algorithm 2-1 when the machines are unrelated, and the extent to which the MILP can be solved with a commercial solver. In all of our experiments with identical machine instances, we use an ordered list (from largest to smallest processing time) in Step ‘a’ of Stage 2 of Algorithm 2-1.

First, we consider an identical and unrelated machine problem instance with $m = 2$, $n_1 = 10$, and $n_2 = 20$. We generate the job processing times for the identical machine instance by simulating 30 observations of $\max\{5,\ \mathcal{N}(30, 15)\}$, i.e., a truncated normal distribution with a minimum value of 5, mean value of 30, and standard deviation of 15. Similarly, we generate the processing times for the unrelated machine instance by simulating two independent sets of 30 observations of $\max\{5,\ \mathcal{N}(30, 15)\}$, one for each machine. We round all simulated processing times to the nearest integer. Finally, we set $\epsilon = 0.0001$.

Figure 2 (a) shows that the schedules generated by Algorithm 2-1 are nearly optimal for the identical machine instances. For the unrelated machine instances (b), we see that the makespan of the schedule generated by Algorithm 2-1 improves in τ as the solution converges to reassigning zero jobs. For the identical machine instances, makespan increases linearly in τ following an inflection at $\tau \approx 100$, after which 2τ always exceeds $\sum_{j \in J_i} p_{ij} x_{ij}$ for all $i \in [m]$ under the optimal policy.

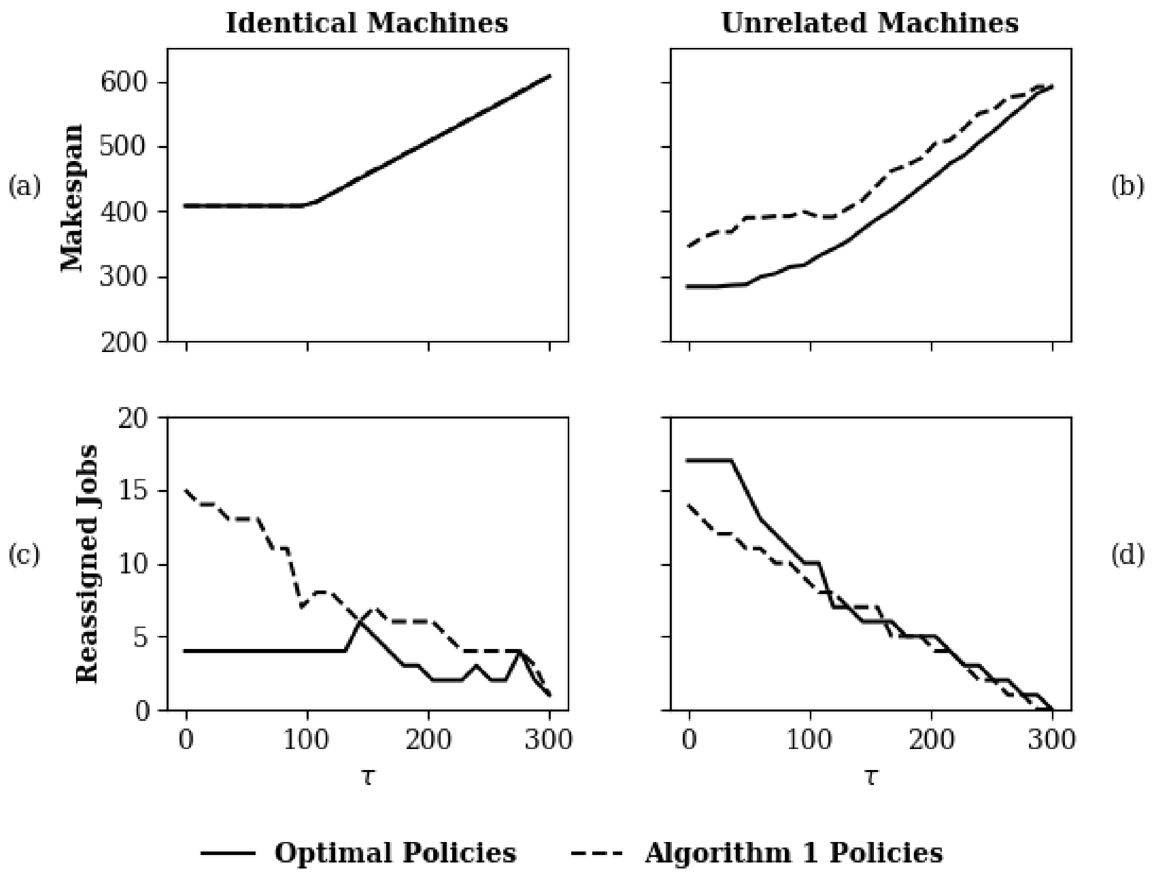


Figure 2: Objective value and number of reassigned jobs as a function of τ for policies generated by Algorithm 2-1 and the MILP defined by (2-2) for a system with 2 machines.

Policies generated by Algorithm 2-1 tend to reassign more jobs than optimal policies in the identical machine case (c), although this difference decreases as the solution converges in τ to reassigning zero jobs. The excessive reassignment is a result of the fact that Algorithm 2-1 does not have a parameter like ϵ that penalizes job reassignment. For the unrelated machine case (d), optimal policies tend to reassign more jobs than policies generated by Algorithm 2-1 because the MILP can better exploit the best machine for each particular job. Algorithm 2-1 Stage 1 also retains a base level (at least 2τ if possible) of work at each machine, whereas the MILP is free to reassign as many jobs as required to minimize makespan.

Note that the number of reassigned jobs under the optimal policy does not necessarily decrease monotonically in τ. We see this behavior because the total duration of reassigned work does not necessarily decrease monotonically in the number of reassigned jobs, e.g., three small jobs can require less processing time than one large job.

Next consider Figure 3, which depicts the empirical approximation ratio ρ of Algorithm 2-1 over 10,000 randomly generated identical-machine and unrelated-machine problem instances. We compute the empirical approximation ratio with the MILP (2-2) and use the following distributions for both the identical and unrelated machine instances: $m \sim \mathcal{U}(2,\ 3)$, $n_i \sim \mathcal{U}(2,\ 10)$, and $p_{ij} \sim \max\{5,\ \mathcal{N}(30, 15)\}$. The distributions from which we generate m and n_i are discrete, and we round each simulated processing time to the nearest integer. For each instance, we set transportation time $\tau = \lceil X \frac{1}{m^2} \sum_{i,j} p_{ij} \rceil$, where $X \sim \mathcal{U}(0,\ 0.5)$. In other words, we set τ to be within 0-50% of the hypothetical average workload if the processing of all jobs could be equally split and shared across machines, recalling that Theorem 2.3.2 admits an improved approximation guarantee when the transportation time contributes to less than $\alpha = \frac{2}{9} + \frac{1}{9m} > \frac{2}{9}$ of the optimal makespan. Finally, we set $\epsilon = 0.0001$ for consistency with the other computational studies. (We could use $\epsilon = 0$ because this example is not concerned with reassignment volume.)

Note that the histograms in Figure 3 (c) and (d) exclude the first bin, i.e., $\rho = 1$; the proportions of instances in which $\rho = 1$ are indicated by the pie charts in (a) and (b). Algorithm 2-1 generates an optimal policy for 24.8% of the identical-machine problem instances and only 3.7% of the unrelated-machine problem instances.

In Figure 4 we examine the efficient frontier between makespan and the number of reassigned jobs, a practical consideration for the decision-maker. We do so by modifying the objective function to include the convex combination of makespan and reassignment volume as follows:

$$\min \left\{ \lambda\mu + (1-\lambda)\,\epsilon \sum_{i\in[m]} \sum_{j\in[n]\setminus J_i} x_{ij} \ :\ \lambda \in [0,1] \right\}. \tag{2-8}$$

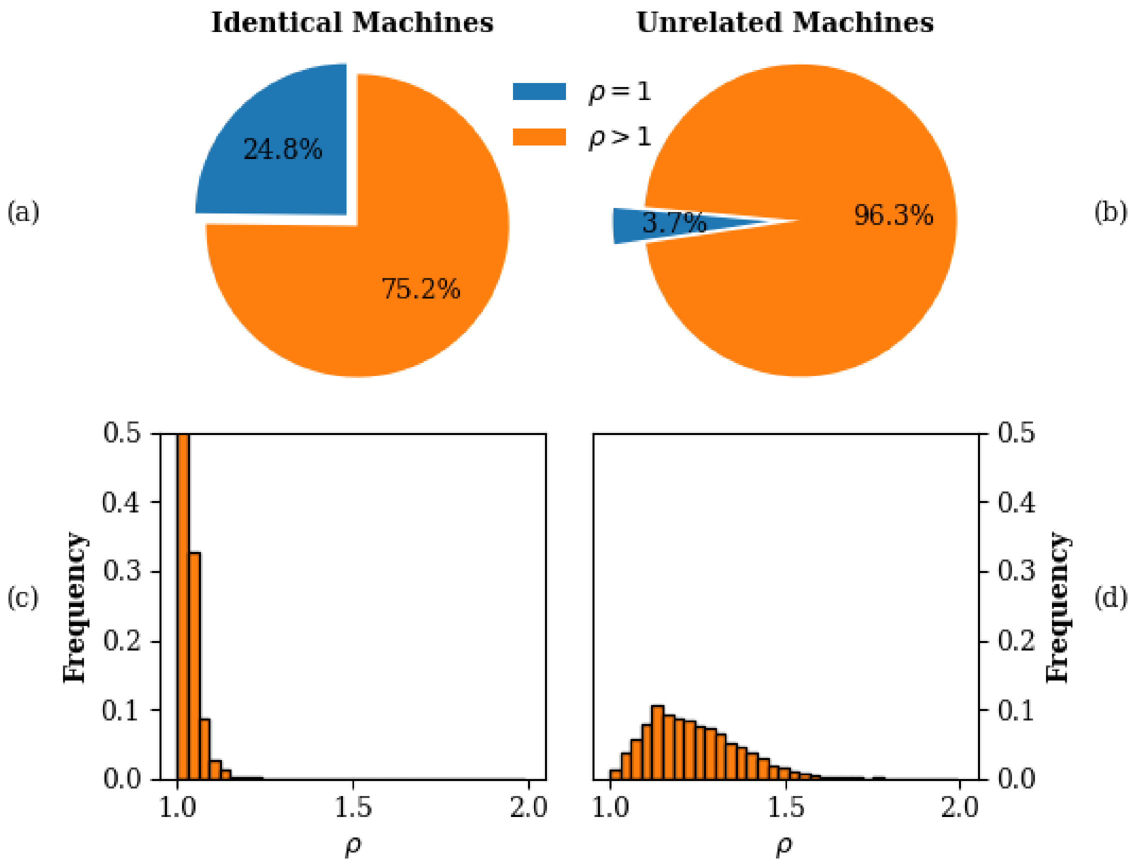


Figure 3: Empirical approximation ratio ρ of Algorithm 2-1 over simulated identical-machine and unrelated-machine problem instances.

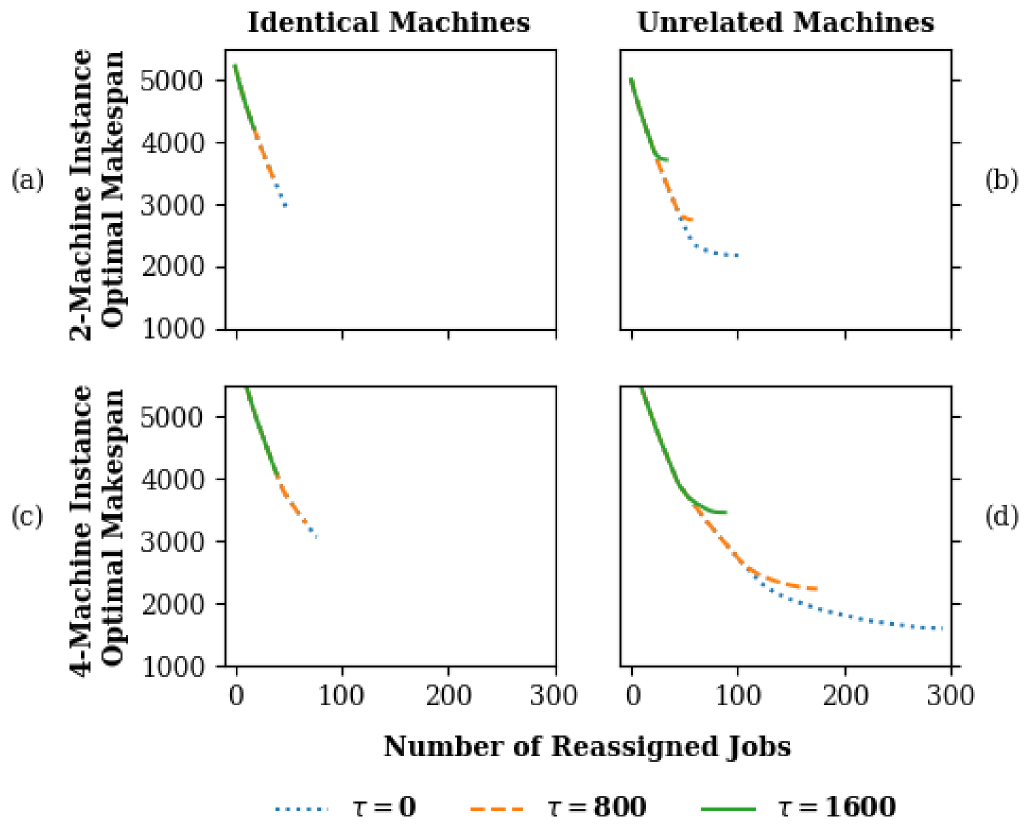


Figure 4: Makespan-reassignment volume efficient frontiers for several problem instances.

In Figure 4 (a) and (b) we consider a 2-machine system with $n_1 = 25$ and $n_2 = 175$. We generate the job processing times for the identical machine instance by simulating 200 random observations of $\max\{5,\ \mathcal{N}(30,15)\}$, and we generate the job processing times for the unrelated machine instance by simulating two independent sets of 200 random observations from $\max\{5,\ \mathcal{N}(30,15)\}$. Similarly, in Figure 4 (c) and (d) we consider a 4-machine system with $n_1 = 25$, $n_2 = 50$, $n_3 = 125$, and $n_4 = 200$. We generate the job processing times by simulating five sets of 400 random observations of $\max\{5,\ \mathcal{N}(30,15)\}$. We round all simulated processing times to the nearest integer, and set $\epsilon = 0.0001$. Then we solve the MILP with the revised objective function (2-8) for three values of τ as indicated in Figure 4.

Figure 4 (b) and (d) illustrate that shorter makespans can be achieved by reassigning more jobs in the unrelated machine cases, compared to the identical machine cases, as the MILP can exploit the fastest machine for any given job. We see that the 4-machine unrelated cases in (d) achieve lower makespans than the 2-machine unrelated cases in (b), despite the fact that job processing times are simulated from the same distribution ($\max\{5,\ \mathcal{N}(30,15)\}$) and $\frac{n}{m} = 100$ in both cases. The lower makespans in (d) occur because the MILP can exploit the fastest machine for any given job, and when there are more machines, there are more opportunities for improved processing times. The unrelated machine cases also converge in reassignment volume to their minimal makespans with pronounced curvature compared to the identical machine cases in (a) and (c), and this curvature is related to the diminishing value of reassigning smaller jobs. Finally, optimal makespan decreases in τ, as expected and consistent with Figure 2.

The efficient frontier for the two identical machine case in Figure 4 (a) appears nearly linear. Recall that the initial workloads in this instance are highly unbalanced ($n_1 = 25$ and $n_2 = 175$). Accordingly, if the optimal policy reassigns one job, then it will tend to be the largest job originally at machine 2, which results in the greatest makespan reduction. And if the optimal policy reassigns two jobs, they will tend to be the largest and second largest jobs of machine 2, and so on until the difference in workloads between the machines is smaller than the largest available job for reassignment, at which point a smaller job(s) is reassigned until makespan cannot be improved further. In the four identical machine case (c), the efficient frontier appears to be approximately piecewise linear. The change in slope occurs, similar to (a), when the difference in workloads between the first and second most loaded machines is smaller than the largest available job for reassignment. After that point, as λ allows more reassignments, the average makespan reduction afforded by each reassignment reduces, i.e., the efficient frontier slope reduces.

The approximate slope of the nearly linear portion of each efficient frontier in Figure 4 represents the average makespan reduction afforded per reassigned job. This slope can be a metric that characterizes the potential value of reassigning jobs, thereby helping the system manager in their decision of whether to pursue reassignment for any given problem instance.

Finally, we investigate the extent to which we can solve the MILP (2-2) with the solver by [38]. We consider identical and unrelated machine settings with 10, 20, 30, 40, and 50 machines, and simulate processing times from $\mathcal{U}\,(10,\ 100)$, consistent with [33] and [27].

For each setting, we start with a problem size in which $n = m$, and randomly define the originating machine of each job using a discrete uniform distribution. For this starting problem size, we generate 10 problem instances with simulated processing times, and simulate transportation times consistent with our computational study of algorithm performance, i.e., $\tau = \lceil X \frac{1}{m^2} \sum\limits_{i,j} p_{ij} \rceil$, where $X \sim \mathcal{U}\,(0,\ 0.5)$. We solve each instance and record the average solution time across the 10 instances.

We repeat this process, at each iteration increasing the total number of jobs to be assigned by five. We terminate the process when (i) we reach a maximum problem size of 150 jobs, or (ii) more than 50% of the instances for a given problem size fail to solve within the Gurobi time limit, which we set to 10 minutes. All computations were performed using Gurobi Version 9.5 with its default optimality tolerance of 1.00E-04, an Intel 3.6 GHz CPU, 32 GB of RAM, and a 64-bit operating system.

Figure 5 depicts the resulting solution times by problem size. It is clear that solution time tends to increase exponentially in the number of jobs to be assigned. It is also clear that larger instances of the unrelated machine setting can be solved within the time limit as compared to the identical machine setting. The difficulty in solving the identical machine problem instances is likely caused by symmetry in the problem structure, a known challenge surveyed by [53].

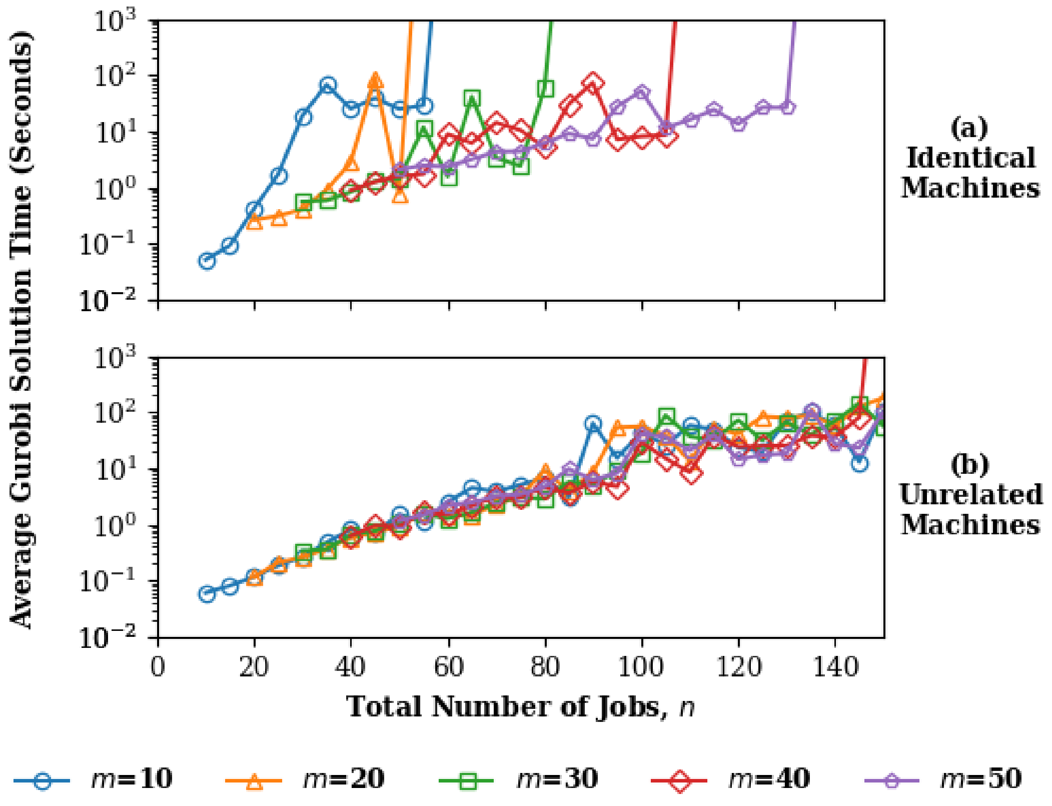


Figure 5: Average Gurobi solution time versus problem size.

2.5 Concluding Remarks

We model and analyze the problem of minimizing makespan across preloaded machines. This problem generalizes the classical parallel machine assignment problem by incorporating machine-specific arrival and delivery times. We develop an MILP formulation of the problem along with a two-stage list scheduling algorithm that generalizes the existing arbitrarily ordered list scheduling algorithm. The need for two stages arises from the fact that existing list scheduling algorithms do not account for the transportation times considered herein. We show that the two-stage list scheduling algorithm is a 2-approximation algorithm when the machines are identical, matching the guarantee of the arbitrarily ordered list scheduling algorithm. We also show that by using lists that are ordered from largest to smallest processing time, instead of arbitrarily ordered lists, our algorithm admits an improved approximation guarantee when there are a *large* number of *small* jobs. Through computational study, we demonstrate that the algorithm often performs much better than its worst-case performance guarantee, even in the unrelated machine setting.

3.0 The Meet-in-the-Middle k-Center Problem

3.1 Introduction

Given client locations and potential facility locations, the goal in *facility location problems*, broadly speaking, is to determine where to build *facilities* in order to serve *clients* in a manner that optimizes some objective. Typically the objective is a function of the distances between the facility and client locations. For instance, given a set of locations and specified distances between the locations, the goal in the k-center problem is to build k facilities at different locations to minimize the maximum distance between a location and its nearest facility. (In this problem the client locations and potential facility locations are the same.)

Most objectives studied in the facility location literature do not take into account the fact that facilities can often send agents to intermediate locations to *meet-in-the-middle* with clients to reduce service distance. Accounting for the ability to meet-in-the-middle is particularly important in *vaccination campaign design* for geographically dispersed populations with limited transportation resources (e.g., low to middle income countries where many travel by walking).

In traditional vaccination campaign design, the goal is to determine where to build healthcare facilities (or clinics) to administer vaccines to patients who travel to the healthcare facilities. However, it is not feasible to vaccinate an entire population with this traditional strategy when the population is geographically dispersed and has limited transportation resources. To improve vaccination rates in these populations, *outreach* is incorporated into vaccination campaign design [39, 40, 50, 91, 92]. Outreach involves health workers transporting vaccines from medical facilities to other intermediate locations, and patients traveling to those intermediate locations to become vaccinated (i.e., facilities and clients meet-in-the-middle). The goal in designing a vaccination campaign that incorporates outreach is therefore threefold: locate healthcare facilities; identify which locations should receive outreach services; and amongst the locations that do not receive outreach services, identify the location to which their clients should travel for vaccination (i.e., a facility itself or a non-facility location that receives outreach, the latter of which constitutes meeting-in-the middle).

In this work we study a variation of the k-center problem in which facilities can send agents to meet-in-the-middle with clients. Our motivation for studying a variation of the k-center problem (as opposed to some other facility location problem) is twofold. First, the k-center problem is quite fundamental and finds many applications (in addition to vaccine campaign design). It follows that it is a natural setting to study the meet-in-the-middle concept from a more general standpoint, which is a main goal of this work. Second, solving large-scale instances of vaccine campaign design problems that arise in practice requires heuristics. A number of heuristics have been developed for vaccine campaign design problems that involve outreach, but to the best of our knowledge, theoretical guarantees have yet to be established for these heuristics. Because there are a number of approximation algorithms for the *metric* k-center problem that are still useful in practice (see the survey [31]), it is natural to consider developing an approximation algorithm for a variation of the k-center problem.

3.1.1 Problem Statement

Suppose that there are n locations labeled with indices $[n] := \{1, \ldots, n\}$. As in the k-center problem, we would like to establish k *facility locations*, but we additionally must decide whether each location without a facility will be a *receiving location* or a *retrieving location.* A receiving location receives goods from agents sent one of the facilities, while a retrieving location sends clients to retrieve goods from a facility location or a receiving location. In the latter case, a facility location sends agents and retrieving location sends clients to *meet-in-the-middle* at a receiving location. If location $i \in [n]$ is a receiving location and location $j \in [n]$ is a facility location, there is a *receiving cost* of $c_{ij} \in \mathbb{Z}_{\geq 0}$ for location i to receive goods from location j. If location i is a retrieving location and location j is a facility or receiving location, there is a *retrieving cost* of $w_{ij} \in \mathbb{Z}_{\geq 0}$ for location i to retrieve goods from location j. We assume throughout that $c_{ii} := 0$ and $w_{ii} := 0$ for each $i \in [n]$. The receiving and retrieving costs should be thought of as the cost to assign a location to receive and retrieve goods, respectively, from another location. In the context of vaccine campaign design, health workers travel from facility locations to receiving locations, and patients travel from retrieving locations to facility or receiving locations to be vaccinated. There is a receiving cost of c_{ij} for health workers to travel from location j to location i, and there is a retrieving cost of w_{ij} for patients to travel from location i to location j. Notice that, in this context, it is not necessarily the case that $c_{ij} = w_{ij}$. For instance, only a few health care workers might to need to drive from location j to location i, while many patients might need to walk from location i to location j.

The goal is to determine facility, receiving, and retrieving locations to minimize the maximum receiving or retrieving cost that must be paid for all locations to obtain goods. This objective is the meet-in-the-middle analogue of the objective of the standard k-center problem. Note that in the special case in which c_{ij} is the distance between location i and location j, and the receiving and retrieving costs are equal (i.e., $c_{ij} = w_{ij}$ for all $i, j \in [n]$), then the goal is to minimize the maximum distance that agents from a facility location and clients from a retrieving location must travel.

Formally, we seek to solve the following optimization problem. For facility locations $F \subseteq [n]$, receiving locations $R_1 \subseteq [n] \setminus F$, and retrieving locations $R_2 = [n] \setminus (F \cup R_1)$, define

$$r_1(F, R_1) := \max_{i \in R_1} \min_{f \in F} c_{if}$$

to be the maximum receiving cost paid, define

$$r_2(F, R_1, R_2) := \max_{i \in R_2} \min_{f \in F \cup R_1} w_{if}$$

to be maximum retrieving cost paid, and define

$$f(F, R_1, R_2) := \max \{r_1(F, R_1), r_2(F, R_1, R_2)\} \tag{3-1}$$

to be maximum receiving or retrieving cost paid. Then we seek to solve the *meet-in-the-middle k-center* problem:

$$\begin{aligned} \min_{F, R_1, R_2 \subset [n]} \quad & f(F, R_1, R_2) \\ \text{s.t.} \quad & |F| = k \\ & R_1 \subseteq [n] \setminus F \\ & R_2 = [n] \setminus (F \cup R_1). \end{aligned} \tag{3-2}$$

It is not hard to see the potential value of accounting for the ability to meet-in-the-middle in facility location. Consider Plot (a) of Figure 6, which depicts $n = 10$ locations on the unit square. Let us take c_{ij} to be the Euclidean distance between location i and location j, and assume that the receiving and retrieving costs are equal (i.e., $c_{ij} = w_{ij}$ for all $i, j \in [n]$). Recall that, in this case, the goal of the meet-in-the-middle k-center problem is to minimize the maximum distance that agents from a facility location and clients from a retrieving location must travel. Plot (b) shows the optimal standard 2-center solution; its objective value (the maximum distance between a location and its nearest facility) is $\eta = 0.52$. Plot (c) shows the facilities from the optimal standard 2-center solution together with the optimal receiving and retrieving locations given the these facility locations (capturing the notion that facilities and clients often meet-in-the-middle in practice even though meeting-in-the-middle is not explicitly considered when determining facility locations). The objective of this solution (i.e., the maximum distance that agents from a facility location and clients from a retrieving location must travel) is $\eta = 0.44$. Plot (d) shows the optimal meet-in-the-middle k-center solution. We observe that the optimal meet-in-the-middle k-center solution yields a smaller maximum distance, namely $\eta = 0.35$, than the former two solutions, capturing the potential value of accounting for the ability to meet-in-the-middle in facility location.

3.1.2 Summary of our Approach and Contributions

We study the meet-in-the-middle k-center problem (3-2) under a variety of different *cost regimes*, i.e., subsets of assumptions about the receiving and retrieving costs. We discuss each of the cost regimes in Section 3.2.

In Section 3.3 we introduce the notion of "distance" d_{ij} between locations $i, j \in [n]$ that depends on the receiving and retrieving costs. The distance d_{ij} can be thought of as the cost at which location i can obtain goods from a facility at location j (possibly by retrieving the goods from another location $k \in [n]$). The notion of distance plays a key role throughout this work. In particular, we show in Section 3.4 that the the meet-in-the-middle k-center problem (3-2) reduces to the k-center problem under this notion of distance. Unfortunately, however, this reduction does not enable us to immediately develop an approximation algorithm for

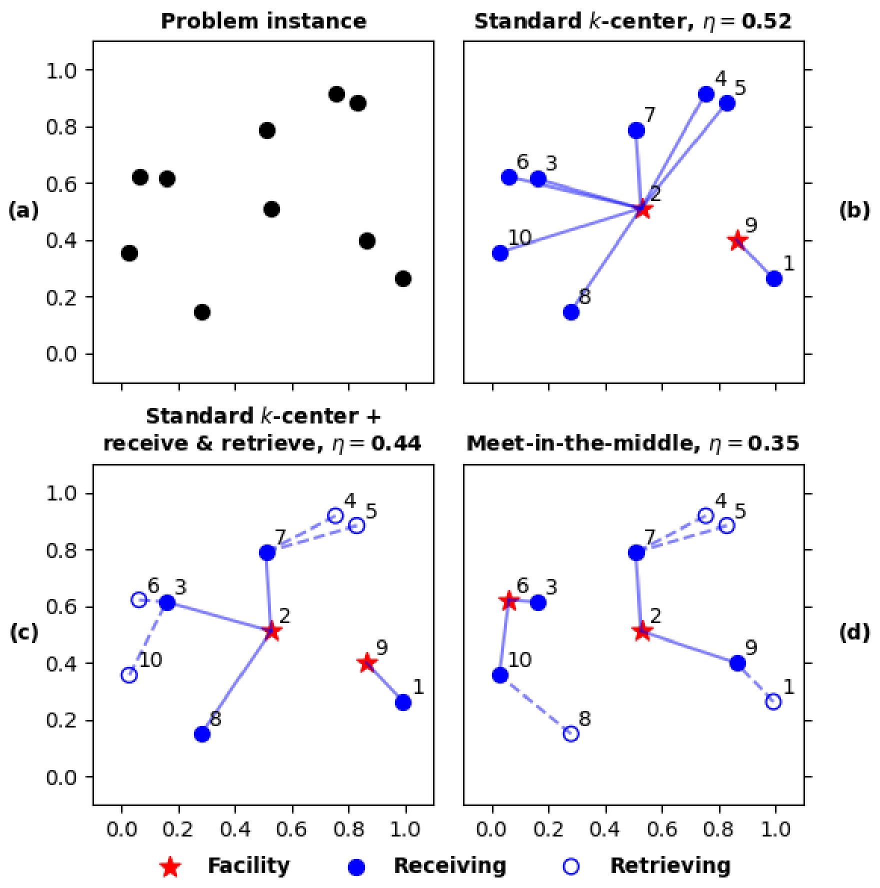


Figure 6: A meet-in-the-middle 2-center problem instance on the unit square together with three feasible solutions. Receiving and retrieving costs equal the Euclidean distances between locations. The quantity η equals the objective value of the solutions. A solid line indicates the facility from which a receiving location obtains goods, and a dashed line indicates the location from which a retrieving location obtains goods.

the meet-in-the-middle k-center problem with well-established techniques for the metric k-center problem because this notion of distance does not necessarily define a metric on $[n]$, even when receiving and retrieving costs individually define metrics on $[n]$. In Section 3.3.1 we show that this notion of distance does not define a metric in any of the cost regimes specified in Section 3.2.

In Section 3.4 we consider the subproblem of determining receiving and retrieving locations given a set of facility locations (recall the solution in Plot (c) of Figure 6); the subproblem also plays a key role throughout this work. We present a polynomial-time algorithm for solving the subproblem in Algorithm 3-1. As a byproduct of the analysis that establishes the correctness of the algorithm, we determine how to reduce the meet-in-the-middle k-center problem (3-2) to the k-center problem under our notion of distance. More specifically, to solve the meet-in-the-middle k-center problem, we establish that we can solve the k-center problem under our notion of distance to obtain optimal facility locations and then apply Algorithm 3-1 (with the optimal facility locations as input) to obtain optimal receiving and optimal retrieving locations.

We propose two mixed-integer linear programming (MILP) formulations for the meet-in-the-middle k-center problem (3-2) in Section 3.5. The first MILP formulation, which we refer to as the *assignment* formulation, follows immediately from the description of the meet-in-the-middle k-center problem. The second formulation, which we refer to as the *k-center* formulation, follows from the reduction established in Section 3.4. Through computational experiments that we present in Section 3.7 we observe that Gurobi solves the latter formulation approximately one order of magnitude faster than the former formulation.

In Section 3.6 we develop an approximation algorithm for the meet-in-middle k-center problem (3-2); the approximation factor depends on the cost regime under consideration. Motivated by the reduction presented in Section 3.4, we consider an algorithm that extends a greedy approximation algorithm for the metric k-center problem. Of course, distance in our setting does not in general induce a metric, so we cannot immediately leverage the existing approximation guarantees. Instead, we present our own analysis that requires a more substantial effort. We also establish that our analysis is tight in Section 3.6.2.

Finally in Section 7 we carry out computational experiments. We investigate the empirical performance of the methods proposed herein, and investigate the advantages of incorporating the ability to meet-in-the-middle versus (i) not meeting-in-the-middle at all and (ii) meeting-in-the-middle but not accounting for this ability when determining facility locations (as in Plot (c) of Figure 6).

3.1.3 Related Literature

The k-center problem. The k-center problem has been extensively studied under the assumption that distances between locations define a metric. Under this assumption, the problem is referred to as the *metric* k-center problem. The metric k-center problem is NP-hard [44]. It is not hard to see that this implies that the meet-in-the-middle k-center problem (3-2) is NP-hard as well. Indeed, an instance of the metric k-center problem with metric d reduces to an instance of the meet-in-the-middle k-center problem with $c = d$ and $w_{ij} = \max_{p,q \in [n]} d_{pq}$ for all $i \neq j \in [n]$. (Here the definition of the retrieving costs w ensures that it is always optimal for a location to receive goods.)

Furthermore, Hochbaum [41] shows that for $\alpha < 2$, it is NP-hard to solve the metric k-center problem to within an approximation factor of α. There are a number of 2-approximation algorithms for the metric k-center problem, including the simple greedy algorithm of Gonzalez [34]. Often this algorithm is referred to as the Gon algorithm. In this work we develop an extension of the Gon algorithm for the meet-in-the-middle k-center problem (3-2). Other 2-approximation algorithms include the Sh algorithm [73] and the HS algorithm [42]. The CDS algorithm proposed in [32] is yet another approximation algorithm, but it has an approximation factor of 3. Nonetheless, it tends to achieve better empirical performance than the Gon, Sh, and HS algorithms.

Fewer works study the setting in which distances do not define a metric. A few studies consider the *asymmetric* k-center problem, namely the k-center problem under the assumption that the metric does not satisfy the symmetry property, but still satisfies the triangle inequality [6, 62, 82]. In particular, [62, 82] propose $O(\log^*(m))$-approximation algorithms, and [6] proposes approximation algorithms with improved approximation factors of $O(\log^*(k))$, where $\log^*$ is the iterated logarithm. In Section 3.4 we show that the meet-in-the-middle k-center problem reduces to a k-center problem with distances that are not necessarily symmetric nor satisfy the triangle inequality.

Many variations of the k-center problem have been considered in the literature; here, we simply discuss the most relevant. A few studies, like this study, consider the possibility of routing through locations to serve clients. Study [45] develops an approximation algorithm for the edge-dilation k-center problem. The goal in the edge-dilation k-center problem is to choose k facility locations to minimize the maximum ratio of geometric to graph distance between a client and its nearest facility. Another related problem is the k-next center problem that [5] first introduced and [52] studied thereafter. The goal in the k-next center problem is to choose k facility locations that minimize the maximum distance between clients and their nearest facility *plus* the distance from this nearest facility to its nearest facility (the backup facility).

Healthcare facility location problems. Healthcare facility location problems have received a considerable amount of research attention; see Daskin and Dean [23] for a review of single-level models, Rahman et al. [67] for a review specific to developing countries, and [4, 68] for more recent reviews.

A number of studies specifically consider vaccine campaign design problems. Recall that in traditional vaccination distribution setups, patients must travel to health facilities to be vaccinated. In low and middle income countries, however, it is extremely difficult to vaccinate certain parts of the population with this type of plan. Part of the difficulty lies in the fact that these countries tend to be geographically dispersed or have nomadic populations. For instance, study [11] demonstrates that vaccination rates in Niger are strongly correlated to patient travel distance, and estimates that only 39% of the population of Niger was within a one-hour walk to a health center during the dry season (and only 24% during the wet season). Similarly, [60] estimates that nearly 40% of the population in Kenya must travel at least one hour to reach government health services. Studies [37, 50, 75, 81] all take into account the fact that requiring patients to travel to a vaccination site negatively affects coverage. In 2002, the World Health Organization and its partners introduced the "reaching every district" (RED) strategy [79] with the goal of improving immunization system performance in areas with low coverage. Outreach trips made from medical facilities to remote locations are an important component of the RED strategy [87]. Outreach trips can involve directly delivering vaccines to a location, or delivering vaccines to a location that patients from other locations then travel to (i.e., meet-in-the-middle).

To the best of our knowledge, study [50] is the first to consider a facility location problem that takes into account outreach trips. The study considers variations of the set covering problem and develops several MILP formulations. All of these formulations aim to maximize location coverage. The work in [39] develops more complicated variations of these set covering models to account for important practical considerations. This study also proposes MILP formulations, but minimizes cost subject to ensuring full coverage instead of maximizing coverage. Subsequent work by [92] considers a variation of the set covering problem that involves vehicle routing and develops an MILP formulation. In a follow up work, [91] considers using machine learning methods to generate approximately optimal solutions to the MILP as a means to rapidly generate a high quality solution, for example, when field conditions necessitate a change to the distribution plan. Most recently, [40] considers a variation of the set covering problem in which the objective is to maximize coverage.

In summary, existing research considers covering-style models and treats travel distance in terms of feasibility. We consider a variation of the k-center problem in which the goal is to minimize the maximum travel distance required to cover all locations.

Other related facility location problems. There is a vast literature on facility location problems; we direct the readers to [14, 70, 71] for comprehensive reviews. The idea of meeting-in-the-middle arises in public services that involve balancing public access with operational efficiency. See [28] for a review of operations research models in urban facility placement. As one example, the placement of public bus stops affects both operational efficiency and public access (see also [57, 72, 84]). While the solution strategies for these problems often address each objective in isolation (i.e., either efficiency or access) [28], the meet-in-the-middle k-center problem (3-2) inherently balances both objectives.

Finally, our problem is related to the multi-level facility location problem (MLFLP); see the survey [61]. Consider, for example, a two-level production-distribution system in which goods are manufactured at production facilities, which then distribute the goods to warehouses, which finally distribute the goods to customers. Our problem is distinct from the MLFLP because each customer in the MLFLP must be assigned to a *sequence* of facility levels; whereas in the meet-in-the-middle k-center problem, customer demand can be satisfied by *either* of the levels individually.

3.1.4 Organization

We introduce and discuss the different cost regimes of interest in Section 3.2. Then in Section 3.3 we introduce and study a notion of distance d_{ij} between locations $i, j \in [n]$ that depends on the receiving and retrieving costs. In Section 3.4 we consider the subproblem of determining receiving and retrieving locations given facility locations. We present a polynomial-time algorithm for solving the subproblem, and we also show how to reduce the meet-in-the-middle k-center problem (3-2) to the k-center problem under our notion of distance. We propose two MILP formulations for the meet-in-the-middle k-center problem (3-2) in Section 3.5. In Section 3.6 we develop an approximation algorithm for the meet-in-middle k-center problem (3-2); the approximation factor depends on the cost regime under consideration. In Section 3.7 we carry out computational experiments. Finally in Section 3.8 we summarize our findings and discuss future research directions.

3.2 Receiving and Retrieving Cost Regimes

We study the facility location problem of interest under different cost regimes. More precisely, we study the problem under different subsets of assumptions about the receiving and retrieving costs; see Assumptions 3.2.1-3.2.3 and Table 1 below. Throughout, the regime under consideration will be clear from context.

Assumption 3.2.1. *Receiving and retrieving costs individually define metrics on* $[n]$*. That is, the costs are symmetric*

$$c_{ij} = c_{ji} \quad \textit{and} \quad w_{ij} = w_{ji}, \quad i, j \in [n],$$

and the costs satisfy the triangle inequality

$$c_{ij} \leq c_{ik} + c_{kj} \quad \textit{and} \quad w_{ij} \leq w_{ik} + w_{kj}, \quad i, j, k \in [n].$$

Recall that, by definition, $c_{ij}, w_{ij} \in \mathbb{Z}_{\geq 0}$ for $i, j \in [n]$, and $c_{ii} = w_{ii} = 0$, and hence we do not need to include these properties in Assumption 3.2.1. We also note that Assumption 3.2.1 is quite natural in light of the fact that the classic k-center problem has been extensively studied under the assumption that distances between locations define a metric on $[n]$.

Assumption 3.2.2. *Receiving and retrieving costs are equal, namely*

$$w_{ij} = c_{ij}, \quad i, j \in [n].$$

From an application standpoint, it may be reasonable to assume that Assumption 3.2.2 holds, for example, if goods are transported from facilities and receiving locations using the same transportation resources that are already at these locations.

Assumption 3.2.3. *For all $i, j, k \in [n]$ such that $i \neq j \neq k$,*

$$c_{ij} \leq \min\{c_{kj} + w_{ik}, c_{ki} + w_{jk}\}.$$

Assumption 3.2.3 states that the cost for location $i \in [n]$ to receive goods from location $j \in [n]$ is less than 1) the total cost for location i to receive goods from location j through meeting at a location $k \in [n]$, and 2) the total cost for location j to receive goods from location i through meeting at a location k.

Note if Assumptions 3.2.1 and 3.2.2 hold, then Assumption 3.2.3 holds. Also note that if Assumption 3.2.1 holds and $c_{ij} \leq w_{ij}$ for all $i, j \in [n]$, then Assumption 3.2.3 holds. We summarize the four different cost regimes of interest (from most general to least general) in Table 1 below.

Table 1: Summary of cost regimes, from most general to least general.

Regime	Description	Defining assumptions
1	No assumptions	None
2	Metric costs	Assumption 3.2.1
3	Related metric costs	Assumption 3.2.1 and 3.2.3
4	Equal metric costs	Assumption 3.2.1 and 3.2.2

3.3 Distance between Locations

In this section we introduce a notion of distance between locations that plays a fundamental role throughout the remainder of the paper. Specifically, we use the notion in Section 3.4 to reduce the meet-in-the-middle k-center problem to a k-center problem, and we further leverage this reduction to develop a MILP formulation in Section 3.5.2 and an approximation algorithm in Section 3.6. In Section 3.3.1 we show that that our notion of distance does not define a metric on $[n]$ in any of the cost regimes described in Section 3.2.

First, for $i, j \in [n]$, we define the *meet-in-the-middle cost*

$$r_{ij} := \begin{cases} \min_{k \in [n] \setminus \{i,j\}} \max\{c_{kj}, w_{ik}\} & i \neq j \\ 0 & i = j. \end{cases}$$

If location i is a retrieving location and location j is a facility location, then r_{ij} is the smallest possible cost (as it would contribute to the objective function) at which location i can obtain goods from a receiving location $k \in [n] \setminus \{i, j\}$ that receives goods from facility location j. Note that r_{ij} is not the *total* meet-in-the-middle cost $c_{kj} + w_{ik}$; instead, r_{ij} is the *maximum* of c_{kj} and w_{ik}. We define the *distance d_{ij} from location i to location j* by

$$d_{ij} := \min\{c_{ij}, w_{ij}, r_{ij}\}. \tag{3-3}$$

The distance d_{ij} is the smallest possible cost at which location i can obtain goods (possibly through a receiving location) from location j. We refer to d_{ij} as the distance between location i and location j because we show in Section 3.4 that the meet-in-the-middle k-center problem reduces to a k-center problem under this notion of distance.

3.3.1 Distance in Different Cost Regimes

Clearly $d_{ij} \in \mathbb{Z}_+$ for $i, j \in [n]$, and $d_{ii} = 0$ for $i \in [n]$, but this notion of distance does not necessarily define a metric in any of the cost regimes described in Section 3.2. It is easy to see that this is the case in the no assumptions cost regime. In the following examples we show that this is also the case for the other three cost regimes of interest. It will be convenient to collect the receiving costs, retrieving costs, and distances into the $n \times n$ matrices C, W, and D, respectively. That is, $C_{ij} = c_{ij}$, $W_{ij} = w_{ij}$, and $D_{ij} = d_{ij}$.

Example 3.3.1 shows that distances in the metric costs regime are not necessarily symmetric, nor do they necessarily satisfy the triangle inequality. Furthermore, the example shows that both symmetry and the triangle inequality can be severely violated.

Example 3.3.1. Suppose that there are $n = 3$ locations, and suppose the receiving and retrieving costs are given by

$$C = \begin{bmatrix} 0 & \epsilon & M+\epsilon \\ \epsilon & 0 & M \\ M+\epsilon & M & 0 \end{bmatrix} \quad \text{and} \quad W = \begin{bmatrix} 0 & M & M+\epsilon \\ M & 0 & \epsilon \\ M+\epsilon & \epsilon & 0 \end{bmatrix},$$

respectively, where $\epsilon, M \in \mathbb{Z}_{\geq 0}$ such that $\epsilon \ll M$. It is easy to check that these costs satisfy Assumption 3.2.1. Furthermore,

$$\begin{aligned}
d_{13} &= \min\{c_{13}, w_{13}, \min\{\max\{c_{23}, w_{12}\}\}\} = \min\{M+\epsilon, M+\epsilon, M\} = M,\\
d_{31} &= \min\{c_{31}, w_{31}, \min\{\max\{c_{21}, w_{32}\}\}\} = \min\{M+\epsilon, M+\epsilon, \epsilon\} = \epsilon,\\
d_{12} &= \min\{c_{12}, w_{12}, \min\{\max\{c_{32}, w_{13}\}\}\} = \min\{\epsilon, M, M\} = \epsilon,\\
d_{23} &= \min\{c_{23}, w_{23}, \min\{\max\{c_{13}, w_{21}\}\}\} = \min\{M, \epsilon, M\} = \epsilon.
\end{aligned}$$

The distances do not satisfy the symmetry condition because $d_{13} = M \gg \epsilon = d_{31}$, and the distances do not satisfy the triangle inequality because $d_{13} = M \gg 2\epsilon = d_{12} + d_{23}$.

Example 3.3.2 shows that distance under the related metric costs regime is not necessarily symmetric, and Example 3.3.3 shows that distance in the equal metric costs regime does not necessarily satisfy the triangle inequality. It follows that distance in the related metric costs regime does not necessarily satisfy the triangle inequality as well. It is straightforward to verify that distance is symmetric in the equal costs regime.

Example 3.3.2. Suppose that there are $n = 3$ locations, and suppose that the receiving and retrieving costs are given by

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \quad \text{and} \quad W = \begin{bmatrix} 0 & 1 & 3 \\ 1 & 0 & 2 \\ 3 & 2 & 0 \end{bmatrix}.$$

It is easy to check that Assumption 3.2.1 and 3.2.3 hold. Observe that

$$d_{13} = \min\{c_{13}, w_{13}, \min\{\max\{c_{23}, w_{12}\}\}\} = \min\{2, 3, 1\} = 1$$
$$d_{31} = \min\{c_{31}, w_{31}, \min\{\max\{c_{12}, w_{32}\}\}\} = \min\{2, 3, 2\} = 2.$$

Hence the symmetry condition does not hold because $d_{13} = 1 \neq 2 = d_{31}$.

Example 3.3.3. Let $x_1 = 0$, $x_2 = 1$, $x_3 = 2$, $x_4 = 4$, and $x_5 = 6$. Also let $c_{ij} = w_{ij} = |x_i - x_j|$ for $i, j \in \{1, \ldots, 5\}$. It follows that Assumption 3.2.1 and 3.2.2 hold. We have that

$$\begin{aligned} d_{15} &= \min\{c_{15}, \min\{\max\{c_{12}, c_{25}\}, \max\{c_{13}, c_{35}\}, \max\{c_{14}, c_{45}\}\}\} \\ &= \min\{6, \min\{5, 4, 4\}\} = 4, \\ d_{13} &= \min\{c_{13}, \min\{\max\{c_{12}, c_{23}\}, \max\{c_{14}, c_{43}\}, \max\{c_{15}, c_{53}\}\}\} \\ &= \min\{2, \min\{1, 4, 6\}\} = 1, \\ d_{35} &= \min\{c_{35}, \min\{\max\{c_{31}, c_{15}\}, \max\{c_{32}, c_{25}\}, \max\{c_{34}, c_{45}\}\}\} \\ &= \min\{4, \min\{6, 5, 2\}\} = 2. \end{aligned}$$

Hence the triangle inequality does not hold because $d_{15} = 4 \nleq 3 = d_{13} + d_{35}$.

Example 3.3.2 and 3.3.3, in contrast to Example 3.3.1, do not demonstrate that symmetry and the triangle inequality can be severely violated. Proposition 3.3.1 below shows that symmetry and the triangle inequality can only be violated up to small multiplicative constants in the related metric costs regime. It follows that the triangle inequality can only be violated up to a small multiplicative constant in the equal metric costs regime as well.

Proposition 3.3.1. *In the related metric costs regime, it holds that*

$$d_{ij} \leq 2d_{ji}, \quad i, j \in [n],$$

and

$$d_{ij} \leq 2(d_{ik} + d_{kj}), \quad i, j, k \in [n].$$

Proof. First we show that $\min\{c_{ji}, w_{ji}\} \leq 2d_{ji}$. For each $\ell \in [k]$, it follows from Assumption 3.2.3 that

$$c_{ji} \leq \min\{c_{\ell i} + w_{j\ell}, c_{\ell j} + w_{i\ell}\} \leq c_{\ell i} + w_{j\ell} \leq 2\max\{c_{\ell i}, w_{j\ell}\},$$

and hence

$$d_{ji} = \min\left\{c_{ji}, w_{ji}, \min_{\ell \in [n]\setminus\{j,i\}} \max\{c_{\ell i}, w_{j\ell}\}\right\} \geq \min\left\{c_{ji}, w_{ji}, \frac{1}{2}c_{ji}\right\} \geq \frac{1}{2}\min\{c_{ji}, w_{ji}\}.$$

Thus, $\min\{c_{ji}, w_{ji}\} \leq 2d_{ji}$. Identical arguments show that $\min\{c_{ik}, w_{ik}\} \leq 2d_{ik}$ and that $\min\{c_{kj}, w_{kj}\} \leq 2d_{kj}$.

Observe that

$$d_{ij} = \min\{c_{ij}, w_{ij}, r_{ij}\} \leq \min\{c_{ij}, w_{ij}\} = \min\{c_{ji}, w_{ji}\} \leq 2d_{ji},$$

where the second equality follows from Assumption 3.2.1.

Next observe that

$$\begin{aligned} d_{ij} &= \min\{c_{ij}, w_{ij}, r_{ij}\} \\ &\leq \min\{c_{ij}, w_{ij}\} \\ &\leq \min\{c_{ik} + c_{kj}, c_{kj} + w_{ik}, c_{ki} + w_{jk}, w_{ik} + w_{kj}\} \\ &= \min\{c_{ik} + c_{kj}, w_{ik} + c_{kj}, c_{ik} + w_{kj}, w_{ik} + w_{kj}\} \\ &\leq \min\{c_{ik}, w_{ik}\} + \min\{c_{kj}, w_{kj}\} \\ &\leq 2(d_{ik} + d_{kj}), \end{aligned}$$

where the second inequality follows from Assumption 3.2.1 and 3.2.3, and the second equality follows from Assumption 3.2.1. □

Remark 3.3.1. Example 3.3.2 establishes that the analysis of Proposition 3.3.1 is tight with regards to symmetry. However, Example 3.3.3 does not establish that the analysis is tight with respect to the triangle inequality. We do not need Proposition 3.3.1 to establish any of the main results in this work, so we do not concern ourselves with further investigating whether or not the analysis is tight.

3.4 Determining Receiving and Retrieving Locations given Facility Locations

Given facility locations $F \subseteq [n]$, we consider the problem of determining which of the remaining locations $[n] \setminus F$ should be receiving and retrieving locations to minimize the maximum cost paid to obtain goods. That is, we consider solving the following subproblem of (3-2):

$$\min_{R_1, R_2} \left\{ f(F, R_1, R_2) : R_1 \subseteq [n] \setminus F \text{ and } R_2 = [n] \setminus (F \cup R_1) \right\}. \tag{3-4}$$

We present an algorithm for solving subproblem (3-4) in Algorithm 3-1; Theorem 3.4.1 establishes that Algorithm 3-1 solves the problem of interest in the no assumptions cost regime (see Section 3.2). As a byproduct of our analysis of Algorithm 3-1, we also show that the the meet-in-the-middle k-center problem reduces in polynomial time to the (not necessarily metric) k-center problem in the no assumptions cost regime; see Remark 3.4.2 below.

Steps 1-3 of Algorithm 3-1 compute the facility location f_i that is closest to each location $i \in [n] \setminus F$ without a facility. Steps 4-14 set a location to be a receiving location if is optimal for the location to receive goods (see Steps 6-7) from a facility or if it is optimal for another location to retrieve goods from the location (see Steps 8-13). Intuitively, we make the location a receiving location in the latter case because if it is optimal for another location to retrieve goods from the location, then the objective value of (3-2) will be at least the cost for the location to receive goods. Finally, in Step 15, the algorithm sets the locations not yet selected to be retrieving locations.

Computational complexity. Algorithm 3-1 takes the meet-in-the-middle costs $\{r_{ij}\}_{i,j \in [n]}$

Algorithm 3-1 Determining receiving and retrieving locations.

Input: Facility locations $F \subset [n]$, receiving costs $\{c_{ij}\}_{i,j\in[n]}$, retrieving costs $\{w_{ij}\}_{i,j\in[n]}$, meet-in-the-middle costs $\{r_{ij}\}_{i,j\in[n]}$, and distances $\{d_{ij}\}_{i,j\in[n]}$

```
 1: for $i \in [n] \setminus F$ do
 2:     $f_i \leftarrow \operatorname{argmin}_{f\in F} d_{if}$
 3: end for
 4: Initialize receiving locations $R_1 \leftarrow \emptyset$
 5: for $i \in [n] \setminus F$ do
 6:     if $d_{if_i} = c_{if_i}$ then
 7:         $R_1 \leftarrow R_1 \cup \{i\}$
 8:     else if $d_{if_i} = r_{if_i}$ then
 9:         $j \leftarrow \operatorname{argmin}_{k\in[n]\setminus\{i,f_i\}} \max\{c_{kf_i}, w_{ik}\}$
10:         if $j \notin F$ then
11:             $R_1 \leftarrow R_1 \cup \{j\}$
12:         end if
13:     end if
14: end for
15: Set retrieving locations $R_2 \leftarrow [n] \setminus (F \cup R_1)$
16: Return $R_1$ and $R_2$
```

and distances $\{d_{ij}\}_{i,j\in[n]}$ as input. Meet-in-the-middle costs require $O(n^3)$ operations to compute, and distances require $O(n^2)$ operations to compute (assuming that the meet-in-the-middle costs have already been computed). Therefore, the total computational complexity of precomputing these costs is $O(n^3)$.

Assuming $|F| = k$, Steps 1-3 require $O((n-k)k)$ operations, and Steps 4-14 require $O((n-k)n)$ operations because Step 9 requires $O(n)$ operations. However, if the quantities in Step 9 are precomputed, which can be done while computing the meet-in-the-middle costs at no additional expense to the big-O computational complexity, then Steps 4-14 only require $O(n-k)$ operations. Therefore, the computational complexity of Algorithm 3-1 with and without precomputing costs is $O((n-k)k)$ and $O(n^3)$, respectively. Consequently, Algorithm 3-1 runs in polynomial time.

Correctness. Theorem 3.4.1 establishes that Algorithm 3.4.1 correctly identifies an optimal solution of (3-4). Note that the theorem does not make any assumptions about the receiving or retrieving costs.

Theorem 3.4.1. *Let $F \subset [n]$, and let R_1 and R_2 be the output of Algorithm 3-1 run with input F. Then R_1 and R_2 are optimal for subproblem* (3-4).

Before proving Theorem 3.4.1, we establish a preliminary result. Specifically, we establish a lower bound on the optimal value of subproblem (3-4); see Lemma 3.4.1. Intuitively, the lower bound holds because d_{if} is the smallest possible cost at which location $i \in [n] \setminus F$ can obtain goods (possibly through a receiving location) from location $f \in F$.

Remark 3.4.1. The proof of Theorem 3.4.1 reveals that the lower bound of Lemma 3.4.1 is actually tight; that is, $\max_{i\in[n]\setminus F} \min_{f\in F} d_{if} = f^*$. We formally make record of this fact in Corollary 3.4.1.

Lemma 3.4.1. *Let $F \subset [n]$. Then*

$$\max_{i\in[n]\setminus F} \min_{f\in F} d_{if} \leq f^*,$$

where f^* is the optimal value of subproblem (3-4).

Proof. Suppose that R_1 and R_2 are optimal for (3-4); that is, $f(F, R_1, R_2) = f^*$. Let $j \in [n] \setminus F = R_1 \cup R_2$. It is sufficient to show that $\min_{f \in F} d_{jf} \leq f^*$.

If $j \in R_1$, then

$$\begin{aligned}
\min_{f \in F} d_{jf} &= \min_{f \in F} \min\{c_{jf}, w_{jf}, r_{jf}\} \\
&\leq \min_{f \in F} c_{jf} \\
&\leq \max_{i \in R_1} \min_{f \in F} c_{if} \\
&= r_1(F, R_1) \\
&\leq \max\{r_1(F, R_1), r_2(F, R_1, R_2)\} \\
&= f^*.
\end{aligned}$$

Accordingly, suppose that $j \in R_2$. Let $\ell \in \operatorname{argmin}_{k \in F \cup R_1} w_{jk}$. We consider two cases:

Case 1. Suppose that $\ell \in F$. Then

$$\begin{aligned}
\min_{f \in F} d_{jf} &= \min_{f \in F} \min\{c_{jf}, w_{jf}, r_{jf}\} \\
&\leq \min_{f \in F} w_{jf} \\
&= \min_{k \in F \cup R_1} w_{jk} \\
&\leq \max_{i \in R_2} \min_{k \in F \cup R_1} w_{ik} \\
&= r_2(F, R_1, R_2) \\
&\leq \max\{r_1(F, R_1), r_2(F, R_1, R_2)\} \\
&= f^*.
\end{aligned}$$

Case 2. Suppose that $\ell \in R_1$. Then we have that

$$\begin{aligned}
\min_{f\in F} d_{jf} &= \min_{f\in F} \min\{c_{jf}, w_{jf}, r_{jf}\} \\
&\le \min_{f\in F} r_{jf} \\
&= \min_{f\in F} \min_{k\in[n]\setminus\{j,f\}} \max\{c_{kf}, w_{jk}\} \\
&\le \min_{f\in F} \max\{c_{\ell f}, w_{j\ell}\} \\
&= \max\left\{\min_{f\in F} c_{\ell f}, w_{j\ell}\right\} \\
&\le \max\left\{\max_{i\in R_1}\min_{f\in F} c_{if}, \max_{i\in R_2}\min_{k\in F\cup R_1} w_{ik}\right\} \\
&= \max\left\{r_1(F,R_1), r_2(F,R_1,R_2)\right\} \\
&= f^*,
\end{aligned}$$

where the last inequality follows from $\ell \in R_1$, $j \in R_2$, and $\ell \in \operatorname{argmin}_{k\in f\cup R_1} w_{jk}$. $\square$

With Lemma 3.4.1 in hand, we are now prepared to prove Theorem 3.4.1.

Proof of Theorem 3.4.1. Let f^* denote the optimal value of (3-4). From Lemma 3.4.1, it is sufficient to show that

$$f(F,R_1,R_2) = \max\{r_1(F,R_1), r_2(F,R_2)\} \le \max_{i\in[n]\setminus F}\min_{f\in F} d_{if}. \tag{3-5}$$

This will also imply that $\max_{i\in[n]\setminus F}\min_{f\in F} d_{if} = f^*$.

First we show that

$$r_1(F,R_1) \le \max_{i\in[n]\setminus F}\min_{f\in F} d_{if}. \tag{3-6}$$

Let $j \in R_1$ and $f_j \in \operatorname{argmin}_{f\in F} d_{jf}$. Algorithm 3-1 either adds j to R_1 in Step 7 or in Step 11. In the former case, since $f_j \in F$,

$$\min_{f\in F} c_{jf} \le c_{jf_j} = d_{jf_j} = \min_{f\in F} d_{jf}, \tag{3-7}$$

where the first equality follows from Step 6. Now let us consider the latter case. From Steps 8-13, there is an index $i \in [n] \setminus F$ such that for $f_i \in \operatorname{argmin}_{f \in F} d_{if}$, it holds that $d_{if_i} = r_{if_i}$ and $j \in \operatorname{argmin}_{k \in [n] \setminus \{i, f_i\}} \max\{c_{kf_i}, w_{ik}\}$. Because $f_i \in F$,

$$\min_{f \in F} c_{jf} \leq c_{jf_i} \leq \max\{c_{jf_i}, w_{ij}\} = r_{if_i} = d_{if_i} = \min_{f \in F} d_{if}, \tag{3-8}$$

where the first equality follows from the definition of r_{if_i}. So, from (3-7) and (3-8), we see that $r_1(F, R_1) = \max_{i \in R_1} \min_{f \in F} c_{if} \leq \max_{i \in R_1} \min_{f \in F} d_{if}$, and consequently (3-6) holds.

Next we show that

$$r_2(F, R_1, R_2) \leq \max_{i \in [n] \setminus F} \min_{f \in F} d_{if}. \tag{3-9}$$

Let $j \in R_2$ and $f_j \in \operatorname{argmin}_{f \in F} d_{jf}$. It follows from Steps 4-15 that either $d_{jf_j} = w_{jf_j}$ or $d_{jf_j} = r_{jf_j}$. In the former case, since $f_j \in F$,

$$\min_{k \in F \cup R_1} w_{jk} \leq w_{jf_j} = d_{jf_j} = \min_{f \in F} d_{jf}. \tag{3-10}$$

Now let us consider the latter case, namely $d_{jf_j} = r_{jf_j}$. From Steps 8-13, there is an index $i \in [n] \setminus F$ such that $i \in \operatorname{argmin}_{k \in [n] \setminus \{j, f_j\}} \max\{c_{kf_j}, w_{jk}\}$, and by Steps 10-12 specifically, either $i \in F$ or $i \in R_1$. Hence, $i \in F \cup R_1$, and

$$\min_{k \in F \cup R_1} w_{jk} \leq w_{ji} \leq \max\{c_{jf_j}, w_{ji}\} = r_{jf_j} = d_{jf_j} = \min_{f \in f} d_{jf}. \tag{3-11}$$

So, from (3-10) and (3-11), it holds that

$$r_2(F, R_1, R_2) = \max_{i \in R_2} \min_{j \in F \cup R_2} w_{ij} \leq \max_{i \in [n] \setminus F} \min_{f \in F} d_{if},$$

implying that (3-9) holds.

The desired result (3-5) follows from (3-6) and (3-9). □

From Lemma 3.4.1 and the proof of Theorem 3.4.1, we obtain:

Corollary 3.4.1. *Let $F \subset [n]$. Then*

$$\max_{i \in [n] \setminus F} \min_{f \in F} d_{if} = f^*,$$

where f^* is the optimal value of subproblem (3-4).

From the above results we see that the meet-in-the-middle k-center problem (3-2) reduces in polynomial time to the k-center problem.

Remark 3.4.2. Consider the following k-center problem under our notion of distance (3-3):

$$\min_{F\subset[n]:|F|=k} \max_{i\in[n]\setminus F} \min_{f\in F} d_{if}. \tag{3-12}$$

From Corollary 3.4.1, it follows that the optimal value of (3-12) equals the optimal value of the meet-in-the-middle k-center problem (3-2). Furthermore, we see from Theorem 3.4.1 that given an optimal solution F^* to (3-12), we can run Algorithm 3-1 with input F^* to obtain $R_1^*, R_2^* \subset [n]$ such that (F^*, R_1^*, R_2^*) is optimal for the meet-in-the-middle k-center problem (3-2). Thus, the meet-in-the-middle k-center problem reduces in polynomial time to a k-center problem. Note, however, that because our notion of distance (3-3) does not necessarily define a metric in any of the cost regimes (see Section 3.2), the reduction does not provide a reduction to the metric k-center problem.

3.5 Integer Programming Formulations

In this section we present two MILP formulations for the meet-in-the-middle k-center problem (3-2). Both formulations are applicable to all four cost regimes identified in Table 1. In Section 3.5.1 we introduce an *assignment formulation* that assigns each receiving location to a facility location and that assigns each retrieving location to a facility or receiving location. And in Section 3.5.2 we introduce a *k-center formulation* based on the reduction of the meet-in-the-middle k-center problem to the k-center problem presented in Remark 3.4.2.

3.5.1 Assignment Formulation

For each $i \in [n]$, we introduce the binary variable $x_i \in \{0,1\}$ to indicate if we set up a facility at location i. And for each $i \neq j \in [n]$, we introduce the binary variable $y_{ij} \in \{0,1\}$

to indicate if location i receives goods from location j, and we introduce the binary variable $z_{ij} \in \{0,1\}$ to indicate if location i retrieves goods from location j.

Consider the following assignment MILP formulation:

$$
\begin{aligned}
\min_{x,y,z,\eta} \quad & \eta \\
\text{s.t.} \quad & \eta \geq c_{ij} y_{ij} && i \neq j \in [n] && \text{(3-13a)} \\
& \eta \geq w_{ij} z_{ij} && i \neq j \in [n] && \text{(3-13b)} \\
& y_{ij} \leq x_j && i \neq j \in [n] && \text{(3-13c)} \\
& z_{ij} \leq x_j + \sum_{\ell \in [n]:\ell \neq j} y_{j\ell} && i \neq j \in [n] && \text{(3-13d)} \\
& x_i + \sum_{j \in [n]:j \neq i} y_{ij} + \sum_{j \in [n]:j \neq i} z_{ij} = 1 && i \in [n] && \text{(3-13e)} \\
& \sum_{i=1}^{n} x_i = k && i \in [n] && \text{(3-13f)} \\
& x \in \{0,1\}^n,\ y, z \in \{0,1\}^{n \times n},\ \eta \in \mathbb{R}.
\end{aligned}
$$

Constraints (3-13a) and (3-13b) ensure that the objective equals the maximum cost paid by a location without a facility. Constraints (3-13c) ensure that location i can only receive goods from location j if location j is a facility location. Constraints (3-13d) ensure that location i can only retrieve goods from location j if location j is a facility or receiving location. Constraints (3-13e) ensure that each location is either a facility, receiving, or retrieving location. Finally, constraints (3-13f) ensure that there are k facility locations.

Note that if (x^*, y^*, z^*, η^*) is an optimal solution to IP formulation (3-13), then (F^*, R_1^*, R_2^*), where

$$
\begin{aligned}
F^* &= \{i \in [n] : x_i^* = 1\}, \\
R_1^* &= \{i \in [n] : y_{ij}^* = 1 \text{ for some } j \neq i \in [n]\}, \text{and} \\
R_2^* &= \{i \in [n] : z_{ij}^* = 1 \text{ for some } j \neq i \in [n]\},
\end{aligned}
$$

is an optimal solution for the meet-in-the-middle k-center problem (3-2).

3.5.2 k-Center Formulation

As in the assignment formulation (Section 3.5.1), for each $i \in [n]$, we let the binary variable $x_i \in \{0,1\}$ indicate whether we set up a facility at location i. And for each $i \neq j \in [n]$, the binary variable $y_{ij} \in \{0,1\}$ indicates whether location i obtains goods from location j. That is, $y_{ij} = 1$ if location j is closest to location i under our notion of distance (3-3).

We formulate the k-center problem (3-12) as the MILP:

$$\begin{aligned}
\min_{x,y,\eta} \quad & \eta \\
\text{s.t.} \quad & \eta \geq \sum_{j\in[n]:j\neq i} d_{ij} y_{ij} && i \in [n] && \text{(3-14a)} \\
& y_{ij} \leq x_j && i \neq j \in [n] && \text{(3-14b)} \\
& x_i + \sum_{j\in[n]:j\neq i} y_{ij} = 1 && i \in [n] && \text{(3-14c)} \\
& \sum_{i\in[n]} x_i = k && i \in [n] && \text{(3-14d)} \\
& x \in \{0,1\}^n,\ y \in \{0,1\}^{n\times n},\ \eta \in \mathbb{R}.
\end{aligned}$$

Constraints (3-14a) enforce that the objective equals the maximum distance from a location without a facility to a location with a facility. Constraints (3-14b) ensure that a location i can only obtain goods from location j if location j is a facility location. Constraints (3-14c) ensure that each location is either a facility location or obtains goods from a facility location. Constraints (3-14d) ensure that there are k facility locations.

Proposition 3.5.1 shows how to construct an optimal solution to the meet-in-the-middle k-center problem (3-2) from an optimal solution to the MILP (3-14). The proposition holds in the no assumptions cost regime and follows immediately from Remark 3.4.2.

Proposition 3.5.1. *Suppose that* (x^*, y^*, η^*) *is an optimal solution to the integer program* (3-14). *Then* (F^*, R_1^*, R_2^*) *is an optimal solution to* (3-2), *where* $R_1^*, R_2^* \subset [n]$ *is the output of Algorithm 3-1 executed with input* $F^* = \{i \in [n] : x_i^* = 1\}$.

3.6 Approximation Algorithm

In this section we present and study a greedy algorithm for approximately solving the meet-in-the-middle k-center problem (3-2); see Algorithm 3-2 below. The algorithm is an extension of the greedy algorithm developed in [34] for the k-center problem.

The algorithm runs as follows. In Steps 1-5, we apply the greedy algorithm of [34], but using our notion of distance (3-3). Specifically, in Step 1, the algorithm initializes the set of facility locations F with an arbitrarily chosen location $i \in [n]$. In Steps 2-5, the algorithm adds new indices to F (according to the rule described in Step 3, which chooses the index of the location that is farthest from the locations in F) until F contains k locations. In Step 6, the algorithm applies Algorithm 3-1 with input F to obtain receiving locations R_1 and retrieving locations R_2. Finally, the algorithm returns (F, R_1, R_2) as a solution to the meet-in-the-middle k-center problem (3-2).

Algorithm 3-2 Greedy algorithm.

Input: Receiving costs $\{c_{ij}\}_{i,j\in[n]}$, retrieving costs $\{w_{ij}\}_{i,j\in[n]}$, meet-in-the-middle costs $\{r_{ij}\}_{i,j\in[n]}$, and distances $\{d_{ij}\}_{i,j\in[n]}$.

```
1: Select any $i \in [n]$ and initialize facility locations $F \leftarrow \{i\}$
2: while $|F| < k$ do
3:     $i \leftarrow \text{argmax}_{j\in[n]\setminus F} \min_{f\in F} d_{jf}$
4:     $F \leftarrow F \cup \{i\}$
5: end while
6: $R_1, R_2 \leftarrow$ Algorithm 3-1 with input $F$, $\{c_{ij}\}_{i,j\in[n]}$, $\{w_{ij}\}_{i,j\in[n]}$, $\{r_{ij}\}_{i,j\in[n]}$, and $\{d_{ij}\}_{i,j\in[n]}$
7: Return $(F, R_1, R_2)$
```

Computational complexity. Recall from Section 3.4 that precomputing the meet-in-the-middle costs and distances requires $O(n^3)$ operations. Step 3 of Algorithm 3-2 requires $O(n\ell)$ operations at the ℓ-th iteration. It follows that, as stated, Steps 2-5 use $O(nk^2)$ operations in total. By using $O(nk)$ memory, this total can be reduced to $O(nk)$ operations. Recall from Section 3.4 that the computational complexity of Algorithm 3-1 with and without precomputing costs is $O((n-k)k)$ and $O(n^3)$, respectively. The computational complexity of Algorithm 3-2 is $O(nk)$ and $O(n^3)$ with and without precomputing, respectively.

Approximation guarantee. The greedy algorithm developed in [34] is a 2-approximation algorithm for the metric k-center problem, but it is not immediately clear to what extent we can establish a similar guarantee for Algorithm 3-2. The main challenge is that distance in our setting does not necessarily define a metric in any of the cost regimes of interest (see Section 3.3.1). While we could utilize the fact that distance approximately defines a metric in some of the cost regimes (see Proposition 3.3.1) to establish an approximation guarantee in these regimes, a more careful analysis yields a better approximation guarantee.

We present our main result in Theorem 3.6.1 below. The theorem states that Algorithm 3-2 is a 3-approximation algorithm for meet-in-the-middle k-center problem (3-2) in the related metric costs regime and a 2-approximation algorithm in the equal metric costs regime.

Theorem 3.6.1. *The following statements hold.*

1. *Algorithm 3-2 is a* 3*-approximation algorithm for the meet-in-the-middle k-center problem* (3-2) *in the related metric costs regime.*
2. *Algorithm 3-2 is a* 2*-approximation algorithm for the meet-in-the-middle k-center problem* (3-2) *in the equal metric costs regime.*

Proof. See Section 3.6.1. □

Proposition 3.6.1 below establishes that the analysis of Theorem 3.6.1 is tight.

Proposition 3.6.1. *The following statements hold.*

1. *There is a meet-in-the-middle k-center problem in the related metric costs regime for which Algorithm 3-2 yields a* 3*-approximation.*
2. *There is a meet-in-the-middle k-center problem in the equal metric costs regime for which Algorithm 3-2 yields a* 2*-approximation.*

Proof. See Section 3.6.2. □

3.6.1 Analysis of Algorithm 3-2

Let (F^*, R_1^*, R_2^*) be an optimal solution to problem (3-2). Define

$$\mathrm{OPT} := f(F^*, R_1^*, R_2^*)$$

to be the optimal value of (3-2). And for each $f \in F^*$, define the sets:

$$\begin{aligned} R_{1f}^* &:= \left\{ i \in R_1^* : f \in \operatorname*{argmin}_{f \in F^*} c_{if} \right\}, \\ R_{2f}^* &:= \left\{ i \in R_2^* : \left[\operatorname*{argmin}_{j \in \{f\} \cup R_{1f}^*} w_{ij} \right] \cap \left[\operatorname*{argmin}_{j \in F^* \cup R_1^*} w_{ij} \right] \neq \emptyset \right\}, \\ C_f &:= \{f\} \cup R_{1f}^* \cup R_{2f}^*. \end{aligned}$$

Notice that if $i \in R_{1f}^* \subset R_1^*$, then it is optimal for receiving location i to receive goods from facility f. Also observe that if $i \in R_{2f}^* \subset R_2^*$, then it is optimal for retrieving location i to retrieve goods from facility location f or a receiving location indexed by R_{1f}^*. We refer to the set C_f as an *optimal cluster*. Note that the optimal clusters cover $[n]$ (i.e., $[n] = \cup_{f \in F^*} C_f$), but two different optimal clusters do not necessarily have a nonempty intersection.

Before proving Theorem 3.6.1, we establish Lemma 3.6.1. The lemma states that if two indices are in the same optimal cluster, then the distance (i.e., our notion of distance) between the two corresponding locations is less than a constant multiple of OPT, where the constant depends on the cost regime.

Lemma 3.6.1. *Let $f \in F^*$ and $i, j \in C_f$. Then*

1. $d_{ji} \leq 3\mathrm{OPT}$ *in the related metric costs regime, and*
2. $d_{ji} \leq 2\mathrm{OPT}$ *in the equal cost metric costs regime.*

Proof. If $i = j$, then the desired result immediately follows because $d_{ij} = 0$. Either $i = f$, $i \in R_{1f}^*$, or $j \in R_{2f}^*$; we consider these three cases below.

Case 1. Suppose that $i = f$. If $j \in R_{1f}^*$, then

$$d_{ji} = d_{jf} \leq c_{jf} \leq \mathrm{OPT}.$$

If $j \in R^*_{2f}$, then it is either optimal for j to retrieve from location f or some location $k \in R^*_{2f}$. In the former case,

$$d_{ji} = d_{jf} \leq w_{jf} \leq \text{OPT},$$

and in the latter case,

$$d_{ji} \leq r_{ji} \leq \max\{c_{ki}, w_{jk}\} \leq \text{OPT}.$$

Case 2. Suppose that $i \in R^*_{1f}$. Either (i) $j = f$, (ii) $j \in R^*_{1f}$, or (iii) $j \in R^*_{2f}$; we consider each of these cases below.

Case 2.1. Suppose that $j = f$. Then

$$d_{ji} = d_{fi} \leq c_{fi} = c_{if} \leq \text{OPT},$$

where the last inequality follows from the fact that $i \in R^*_{1f}$.

Case 2.2. Suppose that $j \in R^*_{1f}$. Then

$$d_{ji} \leq c_{ji} \leq c_{jf} + c_{fi} = c_{jf} + c_{if} \leq 2\text{OPT},$$

where the last inequality follows from the fact that $i, j \in R^*_{1f}$.

Case 2.3. Suppose that $j \in R^*_{2f}$. It is either optimal for location j to retrieve from location f, from location i, or from some location $k \neq i \in R^*_{1f}$. In the first case, we see that

$$d_{ji} \leq r_{ji} \leq \max\{c_{fi}, w_{jf}\} = \max\{c_{if}, w_{jf}\} \leq \text{OPT},$$

where the last inequality follows from $i \in R^*_{1f}$ and the assumption that it is optimal for location j to retrieve from location f. In the second case, it holds that

$$d_{ji} \leq w_{ji} \leq \text{OPT}.$$

And in the last case, we have that

$$\begin{aligned} d_{ji} \leq r_{ji} \leq \max\{c_{ki}, w_{jk}\} &\leq \max\{c_{kf} + c_{fi}, w_{jk}\} = \max\{c_{kf} + c_{if}, w_{jk}\} \\ &\leq \max\{2\text{OPT}, \text{OPT}\} = 2\text{OPT}, \end{aligned}$$

where the last inequality follows from the fact that $i, k \in R^*_{1f}$ and the assumption that it is optimal for location j to retrieve from location k.

Case 3. Suppose that $i \in R^*_{2f}$. Either (*i*) $j = f$, (*ii*) $j \in R^*_{1f}$, or (*iii*) $j \in R^*_{2f}$; we consider these three cases below.

Case 3.1. Suppose that $j = f$. It is either optimal for location i to retrieve from location $j = f$ or from some location $k \in R^*_{1f}$. In the former case,

$$d_{ji} = d_{fi} \leq w_{fi} = w_{if} \leq \text{OPT},$$

and in the latter case,

$$d_{ji} = d_{fi} \leq c_{fi} \leq c_{kf} + w_{ik} \leq 2\text{OPT},$$

where the last inequality follows from the fact that $k \in R^*_{1f}$ and it is optimal for location i to retrieve from location k.

Case 3.2. Suppose that $j \in R^*_{1f}$. It is either optimal for location i to retrieve from location f, from location j, or from some location $k \neq j \in R^*_{1f}$. In the first case, we have that

$$d_{ji} \leq c_{ji} \leq c_{fj} + w_{if} = c_{jf} + w_{if} \leq 2\text{OPT},$$

where the last inequality follows from the fact $j \in R^*_{1f}$ and it is optimal for location i to retrieve from location f. In the second case, we see that

$$d_{ji} \leq w_{ji} = w_{ij} \leq \text{OPT}.$$

And in the final case, we obtain a different approximation, depending on the cost regime. In the related metric costs regime,

$$d_{ji} \leq c_{ji} \leq c_{jf} + c_{fi} \leq c_{jf} + c_{kf} + w_{ik} \leq 3\text{OPT},$$

where the last inequality follows from the fact that $j, k \in R^*_{1f}$ and it is optimal for location i to retrieve from location k. And in the equal costs regime,

$$\begin{aligned} d_{ji} &\leq r_{ji} \leq \max\{c_{fi}, c_{jf}\} \leq \max\{c_{fk} + c_{ki}, c_{jf}\} = \max\{c_{kf} + c_{ik}, c_{jf}\} \\ &\leq \max\{2\text{OPT}, \text{OPT}\} = 2\text{OPT}, \end{aligned}$$

where the last inequality follows from the fact that $j, k \in R^*_{1f}$ and it is optimal for location i to retrieve from location k.

Case 3.3. Suppose that $j \in R^*_{2f}$. It is either optimal for location i to retrieve from location f or from some location $k \in R^*_{1f}$. First let us consider the case in which it is optimal for location i to retrieve from location f. In this case, it is either optimal for location j to retrieve from location f as well or from some location $\ell \in R^*_{1f}$. In the former case,

$$d_{ji} \leq w_{ji} \leq w_{jf} + w_{fi} = w_{jf} + w_{if} \leq 2\text{OPT},$$

and in the latter case,

$$\begin{aligned} d_{ji} &\leq r_{ji} \leq \max\{c_{\ell i}, w_{j\ell}\} \leq \max\{c_{\ell f} + w_{fi}, w_{j\ell}\} = \max\{c_{\ell f} + w_{if}, w_{j\ell}\} \\ &\leq \max\{2\text{OPT}, \text{OPT}\} = 2\text{OPT}, \end{aligned}$$

where the last inequality follows from the fact that $\ell \in R^*_{1f}$ and it is optimal for location i and j to retrieve from location f and ℓ, respectively.

Next we consider the case in which it is optimal for location i to retrieve from some location $k \in R^*_{1f}$. It is either optimal for location j to retrieve from location k or some location $\ell \in R^*_{1f}$. In the former case,

$$d_{ji} \leq w_{ji} \leq w_{jk} + w_{ki} = w_{jk} + w_{ik} \leq 2\text{OPT}.$$

Now let us consider the latter case in which it is optimal for location j to retrieve from location ℓ. We obtain a different approximation guarantee, depending on the cost regime. In the related metric costs regime,

$$\begin{aligned} d_{ji} &\leq r_{ji} \leq \max\{c_{\ell i}, w_{j\ell}\} \leq \max\{c_{\ell f} + c_{fi}, w_{j\ell}\} \leq \max\{c_{f\ell} + c_{kf} + w_{ik}, w_{j\ell}\} \\ &\leq \max\{3\text{OPT}, \text{OPT}\} = 3\text{OPT}, \end{aligned}$$

where the last inequality follows from the fact that $k, \ell \in R^*_{1f}$ and that it is optimal for location i and j to retrieve from location k and ℓ, respectively. Now, in the equal cost regime,

$$\begin{aligned} d_{ji} \leq r_{ji} \leq \max\{c_{fi}, c_{jf}\} &\leq \max\{c_{fk} + c_{ki}, c_{j\ell} + c_{\ell f}\} = \max\{c_{kf} + c_{ik}, c_{j\ell} + c_{\ell f}\} \\ &\leq \max\{2\text{OPT}, 2\text{OPT}\} = 2\text{OPT}, \end{aligned}$$

where the last inequality follows from the fact that $k, \ell \in R^*_{1f}$ and that it is optimal for locations i and j to retrieve from locations k and ℓ, respectively. □

We are now prepared to prove Theorem 3.6.1. The proof follows in a similar spirit to the proof of [34] (although our proof requires Lemma 3.6.1).

Proof of Theorem 3.6.1. Let (F, R_1, R_2) denote the output of Algorithm 3-2. From Corollary 3.4.1, it holds that $f(F, R_1, R_2) = \max_{j \in [n] \setminus F} \min_{f \in F} d_{jf}$. Let $j \in [n] \setminus F$. It is then sufficient to show that

$$\min_{f \in F} d_{jf} \leq \alpha \text{OPT}, \tag{3-15}$$

where $\alpha = 3$ or $\alpha = 2$, depending on the cost regime.

Because the optimal clusters cover $[n]$, it holds that $j \in C_f$ for some $f \in F^*$. We consider two cases:

Case 1. Suppose that no two indices in F belong to the same optimal cluster. Then there is an index $i \in F \cap C_f$. Observe that

$$\min_{f \in F} d_{jf} \leq d_{ji} \leq \alpha \text{OPT},$$

where the last inequality follows from Lemma 3.6.1. Thus (3-15) holds.

Case 2. Suppose that two indices in F are in the same optimal cluster, say $k, \ell \in \mathcal{C}_{\hat{f}}$ for some $\hat{f} \in F^*$. Without loss of generality, suppose that Algorithm 3-2 added k to F before ℓ, and let $\hat{F}$ denote the set of indices that were added to F before ℓ. Because $\hat{F} \subseteq F$, we have that

$$\min_{f \in F} d_{jf} \leq \min_{f \in \hat{F}} d_{jf} \leq d_{\ell k} \leq \alpha \text{OPT},$$

where the second inequality follows from Step 3 of Algorithm 3-2, and the last inequality from Lemma 3.6.1. □

3.6.2 Tight Meet-in-the-Middle k-Center Instances

To establish Proposition 3.6.1, we consider simple instances of (3-2) (instead of an infinite family of instances) to provide clear insights into the properties that make an instance challenging for Algorithm 3-2.

Proof of Proposition 3.6.1. Let us establish the first statement of the proposition. Consider a meet-in-the-middle 1-center instance with the following receiving and retrieving costs, respectively.

$$C = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \quad \text{and} \quad W = \begin{bmatrix} 0 & 1 & 2 & 5 \\ 1 & 0 & 1 & 4 \\ 2 & 1 & 0 & 3 \\ 5 & 4 & 3 & 0 \end{bmatrix}.$$

It is straightforward to verify that Assumption 3.2.1 holds. Furthermore, Assumption 3.2.3 holds because Assumption 3.2.1 holds and $C \leq W$. Thus, the assumptions of the related metric costs regime are satisfied. Suppose that we initialize Algorithm 3-2 in Step 1 with location $i = 1$. Because $k = 1$, the algorithm then proceeds to Step 6. It is straightforward to verify that, in Step 6, Algorithm 3-1 returns receiving locations $R_1 = \{2, 4\}$ and retrieving locations $R_2 = \{3\}$. Thus Algorithm 3-2 returns the solution $F = \{1\}$, $R_1 = \{2, 4\}$, $R_2 = \{3\}$. Because $c_{21} = 1$, $r_{31} = 1$, and $c_{41} = 3$, it holds that the objective value of this solution equals 3. Now consider the solution $F^* = \{3\}$, $R_1 = \{2, 4\}$, and $R_2 = \{1\}$. Because $c_{23} = 1$, $c_{43} = 1$, and $r_{13} = 1$, it holds that the objective value of this is solution is 1. Therefore, the approximation ratio of the solution of Algorithm 3-2 is 3.

Now let us establish the second statement of the proposition. Consider a meet-in-the-middle 2-center instance with

$$c_{ij} = w_{ij} = |i - j|, \quad i, j \in \{1, \ldots, 7\}.$$

Clearly the assumptions of the equal metric cost regime are satisfied. Suppose that we initialize Algorithm 3-2 in Step 1 with $i = 1$. In Steps 2-4, the algorithm either adds index 6 to F or index 7 because $d_{21} = 1$, $d_{31} = 1$, $d_{41} = 2$, $d_{51} = 2$, $d_{61} = 3$, and $d_{71} = 3$. Suppose that the algorithm adds index 7 to F. It is straightforward to verify that, in

Step 6, Algorithm 3-1 can return receiving locations $R_1 = \{2,3,5,6\}$ and retrieving locations $R_2 = \{4\}$. Thus Algorithm 3-2 can return the solution $F = \{1,7\}$, $R_1 = \{2,3,5,6\}$, and $R_2 = \{4\}$. The objective value of this solution equals 2. Now consider the solution $F^* = \{3,5\}$, $R_1^* = \{2,4,6\}$, and $R_2^* = \{1,7\}$. The objective value of this solution is 1. Therefore, the approximation ratio of the solution of Algorithm 3-2 is 2. □

3.7 Computational Experiments

In this section we present computational experiments to (*i*) investigate the empirical performance of solution methods for the meet-in-the-middle k-center problem and (*ii*) understand the extent to which accounting for the ability to meet-in-the-middle can be advantageous. Throughout this section we consider the equal metric costs regime (see Section 3.2).

First, we investigate the empirical performance of the methods for the meet-in-the-middle k-center problem developed herein. To this end, we (*i*) compare the objective value of approximate solutions from Algorithm 3-2 with the optimal objective value (computed using either MILP formulation (3-13) or (3-14)); we (*ii*) compare the runtime of MILP formulations (3-13) and (3-14); and we (*iii*) study the policy structure of approximate solutions from Algorithm 3-2 and optimal solutions from the MILPs.

Second, we seek to understand the extent to which accounting for the ability to meet-in-the-middle can be advantageous compared to (*i*) not accounting for the ability at all or to (*ii*) "partially" accounting for the ability. In the former case, we compare the performance of meet-in-the-middle k-center solutions with standard k-center solutions. In the latter case, we compare the performance of meet-in-the-middle k-center solutions with solutions whose facility locations are obtained from standard k-center solutions and whose receiving and retrieving locations are obtained from Algorithm 3-1 (with the k-center facility locations as input). This latter comparison provides insight into settings in which meeting in the middle naturally happens but is not explicitly accounted for when determining facility locations.

We study the following exact and approximate solution methods. For clarity, we italicize each method name as we refer to it throughout this section.

- **Exact.** The *exact* method outputs an optimal solution to the meet-in-the-middle k-center problem (3-2). We compute optimal solutions with either MILP formulation (3-13) or (3-14). (In the latter case, we must also apply Algorithm 3-1 to obtain receiving and retrieving locations.)
- **Algorithm 3-2.** *Algorithm 3-2* outputs an approximate optimal solution to the meet-in-the-middle k-center problem (3-2).
- **Exact k-center.** The *exact k-center* method outputs the optimal solution to the k-center problem (i.e., just facility locations) under distances $\{c_{ij}\}_{i,j\in[n]}$. To compute these solutions, we solve the MILP formulation (3-14) with parameters $\{c_{ij}\}_{i,j\in[n]}$ instead of $\{d_{ij}\}_{i,j\in[n]}$.
- **Exact k-center + Algorithm 3-1.** The *exact k-center + Algorithm 3-1* method first runs the *exact k-center* method to obtain facility locations F and then runs Algorithm 3-1 with input F to obtain receiving and retrieving locations. Note that this method is different from applying MILP formulation (3-14) together with Algorithm 3-1 because MILP formulation (3-14) uses our notion of distance (3-3) that accounts for the ability to meet-in-the-middle.
- **Approximate k-center.** The *approximate k-center* method uses the greedy algorithm of [34] to output an heuristic solution to the standard k-center problem (i.e., just facility locations) under distances $\{c_{ij}\}_{i,j\in[n]}$.
- **Approximate k-center + Algorithm 3-1.** The *approximate k-center + Algorithm 3-1* method first runs the *approximate k-center* method to obtain facility locations F and then runs Algorithm 3-1 with input F to obtain receiving and retrieving locations. Note that this method is different from Algorithm 3-2 because Algorithm 3-2 uses our notion of distance (3-3) that accounts for the ability to meet-in-the-middle.

In Section 3.7.1 we consider randomly drawing locations from the unit square and using Euclidean distance for receiving and retrieving costs. Then in Section 3.7.2 we take cities in the United States as locations and use distance between the cities (determined from longitude and latitude coordinates) as receiving and retrieivng costs.

Hardware and software. Our computational study is performed on an LG Gram laptop with a single 11^{th} generation Intel core (2.92 GHz), 16 GB of RAM, and a 64 bit Windows operating system. All experiments are run sequentially. The MILP formulations are solved using Gurobi Optimizer 10.0, and our algorithms are coded using Python 3.9 and the Anaconda 2021.11 distribution.

3.7.1 The Euclidean Meet-in-the-Middle k-Center Problem on the Unit Square

We generate Euclidean meet-in-the-middle k-center problem instances on the unit square as follows. First we draw n points (labeled by $[n]$) from the uniform distribution on the unit square $[0,1]^2$. Next we set the receiving cost c_{ij} for each $i, j \in [n]$ to be the Euclidean distance between points i and j. Recall that throughout this section we consider the equal metric costs regime; accordingly, the retrieving costs are $w_{ij} = c_{ij}$ for each $i, j \in [n]$.

Runtime of MILP formulations (3-13) **and** (3-14)**.** First we compare the runtimes of our two *exact* methods, namely the MILP formulations (3-13) and (3-14). Note that, for the latter case, we do not report the runtime of Algorithm 3-1, which computes receiving and retrieving locations, as it is negligible compared to the MILP runtime for moderately large values of n. We consider Euclidean meet-in-the-middle k-center instances with $n \in \{50, 100, \ldots, 500\}$ and $k/n \in \{0.1, 0.5, 0.9\}$. For each value of n and k/n, we generate 10 instances. We report the corresponding average runtimes, along with their standard deviations, in minutes in Figure 7.

We make the following observations:

- Figure 7 illustrates that the runtime required to solve the *exact* k-center MILP is typically 90% less than the time required to solve the *exact* assignment MILP.
- Runtimes for both methods appear to be exponential in problem size, which is not surprising in light of the fact that the meet-in-the-middle k-center problem is NP-hard. We were only able to solve the *exact* assignment MILP (3-13) for problems sizes up to $n = 500$ within a 15 minute time limit. The right panel in Figure 7, and our experiment in Section 3.7.2, both show that we can solve significantly larger *exact* k-center MILP (3-14) formulations within this time limit.

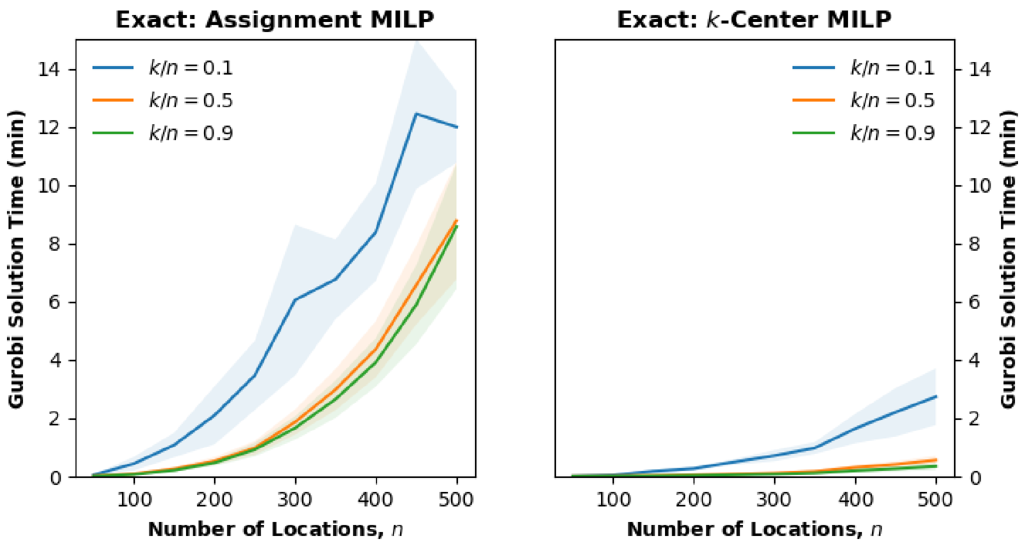
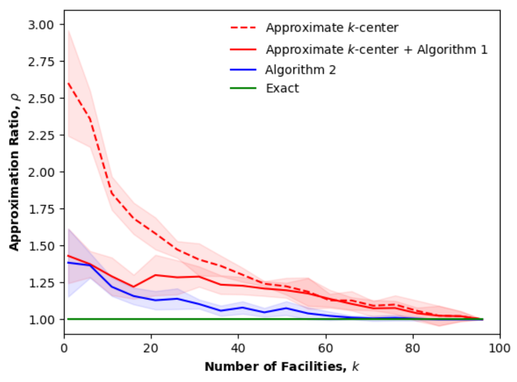


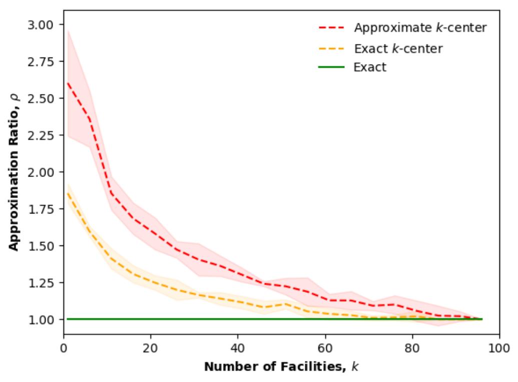
Figure 7: Average solution time of the *exact* assignment MILP (3-13) and *exact* k-center MILP (3-14) for a range of Euclidean meet-in-the-middle k-center problems on the unit square.

- Figure 7 also indicates that problems with fewer facilities are more difficult to solve.

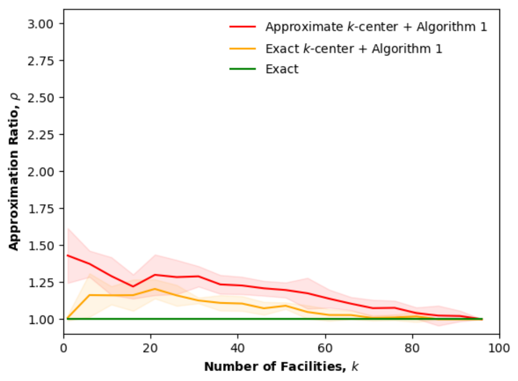
Empirical approximation ratios and the value of meeting in the middle. In our next experiment, we study the empirical approximation ratios of the six methods of interest. We generate Euclidean meet-in-the-middle k-center instances on the unit square with $n = 100$ locations. We consider instances with $k \in \{5, 10, \ldots, 95\}$. For each value of k, we generate 10 instances. We report the sample mean and standard deviation for the empirical approximation ratio $\rho = z^{alg}/z^*$ of each method, where z^{alg} denotes the objective value of an algorithm-generated policy, and z^* denotes the optimal objective value generated by the *exact* assignment MILP formulation (3-13). The plots in the Figure 8 are organized as follows. Plot (a) compares the empirical performance of the following approximation methods: *approximate k-center*, *approximate k-center + Algorithm 3-1*, and *Algorithm 3-2*. Plot (b) illustrates the performance of the following methods that do not consider the ability to meet in the middle at all: *approximate k-center* and *exact k-center*. Plot (c) illustrates



(a) Approximation methods for the meet-in-the-middle k-center problem.



(b) Methods that do not consider the ability to meet in the middle.



(c) Methods that consider the ability to meet in the middle.

Figure 8: Approximation ratios, $\rho = z^{alg}/z^*$, for each method over a range of synthetic problem instances on the Euclidean unit square.

the performance of the following methods that partially consider the ability to meet in the middle: *approximate k-center + Algorithm 3-1* and *exact k-center + Algorithm 3-1.* We make the following observations:

- Plot (a) in Figure 8 supports Theorem 3.6.1, which establishes that *Algorithm 3-2* is a 2-approximation algorithm for the meet-in-the-middle k-center problem in the equal metric costs regime (recall that this experiment was designed with equal costs).

- Plot (a) in Figure 8 also demonstrates that, on average, *Algorithm 3-2* yields better approximate solutions to the meet-in-the-middle k-center problem than *approximate k-center* method and *approximate k-center method + Algorithm 3-1*.
- Comparing Plots (b) and (c) in Figure 8, we see that meeting-in-the-middle yields lower costs than either not meeting-in-the-middle at all, or only partially accounting for the ability to meet in the middle. Naturally the difference in performance is more significant in the former case.

3.7.2 A Meet-in-the-Middle k-Center Problem on the United States Map

In this section we apply the six algorithms to real geospatial data [7] that includes latitude and longitude coordinates for 1,000 cities in the United States. We treat the 1,000 cities as locations, and we measure the distance between two cities by the geospatial distance between their latitude and longitude coordinates.

Figure 9 depicts an optimal solution to the meet-in-the-middle k-center problem (3-2) for $k = 50$. Red stars indicate facility locations, solid blue circles indicate receiving locations, and hollow blue circles indicate retrieving locations. A solid line between two locations indicates that one location receives goods from the other, and dashed lines indicate that one location retrieves goods from the other. Note that we obtained the optimal solution from the *exact* k-center MILP (3-14) within 15 minutes. We could only solve the *exact* assignment MILP (3-13) for instances up to $n = 500$ locations and $k = 75$ facilities within a 15 minute time limit.

Facilities in Figure 9 are distributed in a grid-like manner. Facilities are not necessarily located at population centers, and it is often optimal to locate facilities *between* population centers. For example, the left panel of Figure 10 depicts the Florida portion of the global policy and shows that it is optimal to locate a facility south of Orlando, which is able to directly serve the populations of Saint Petersburg and Port Saint Lucie, and indirectly serve the major populations south of those locations. The right panel of Figure 10 shows that it is optimal to locate a facility near the relatively small population center of Schenectady, New York, which is able to directly and indirectly cover the entire Boston and New York City areas. These insights are not surprising given that our algorithm does not consider population size.

For each of the six methods, Figure 11 depicts the Florida portion of the global policy generated with $n = 1,000$ cities in the United States and $k = 50$ facilities. The *exact* method generated a policy in which the vast majority of the state is covered by only one facility, whereas the other methods required two facilities to cover most of the state. The figure also clearly shows the extended coverage that a given facility can achieve by using the meet-in-the-middle strategy.

Figure 12 plots the approximation ratios of each method over a range of k values for the real geospatial dataset, similar to Figure 8. While we observe the same relative performance between each method as we did in the synthetic data experiments, the methods overall performed better (ρ values closer to 1) in the experiments with real data. The performance degradation observed in the synthetic data experiments is likely because the facility location step is more challenging when the locations are randomly distributed; whereas in real geospatial data, the locations tend to naturally cluster (due to human settlement patterns), and this makes facility location decisions somewhat easier.

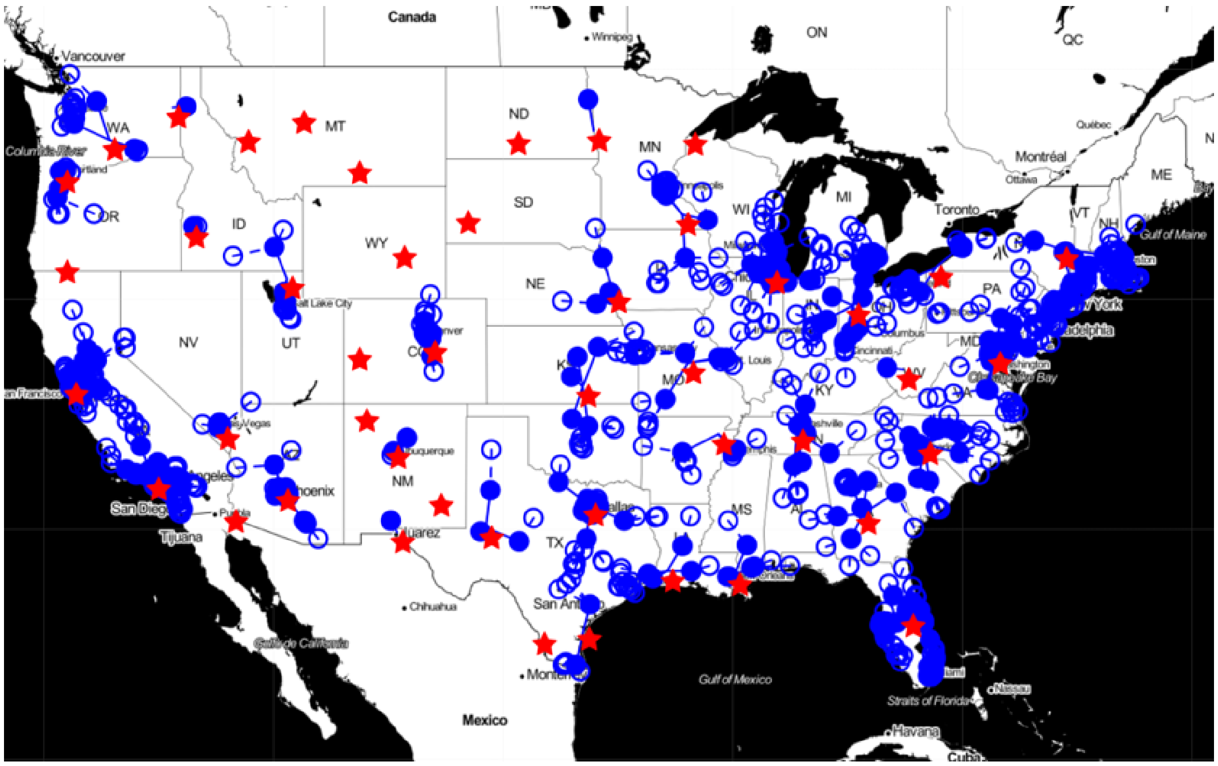


Figure 9: Optimal policy generated by the *exact* method for $n = 1,000$ cities in the United States and $k = 50$ facilities.

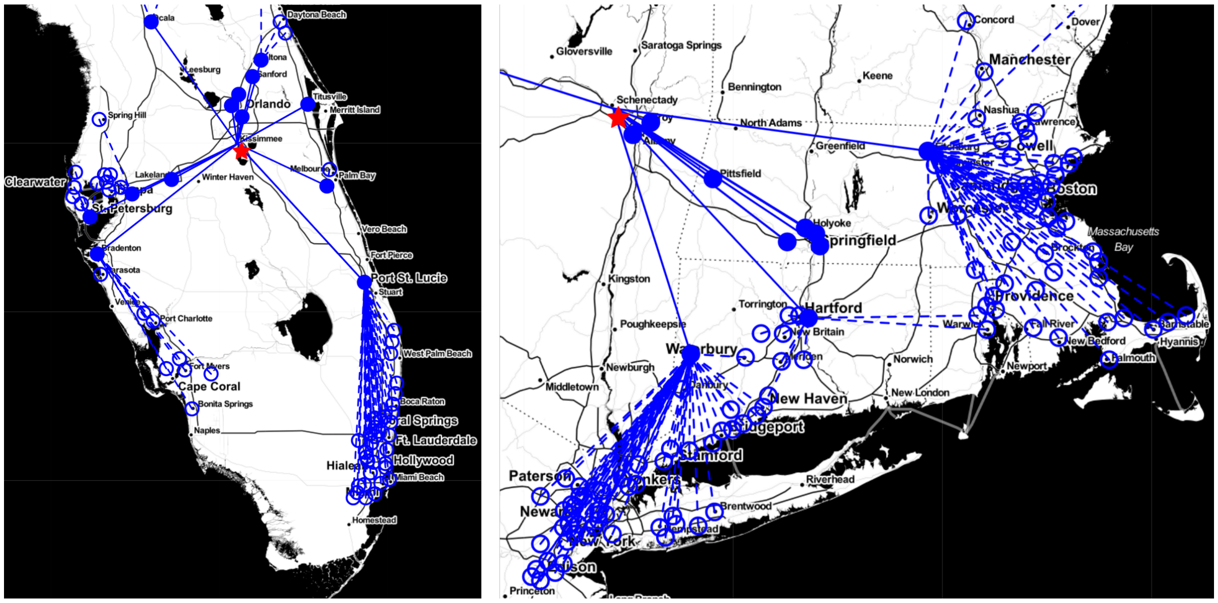
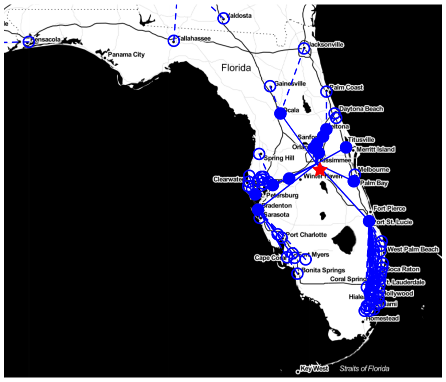
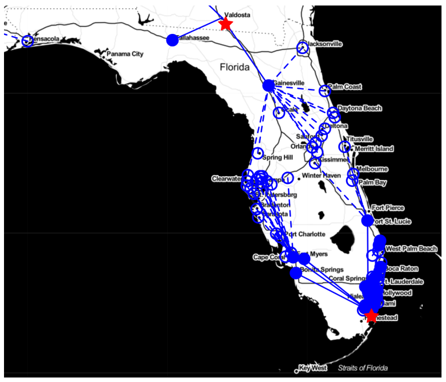
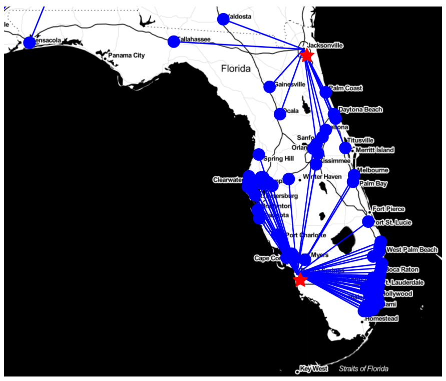
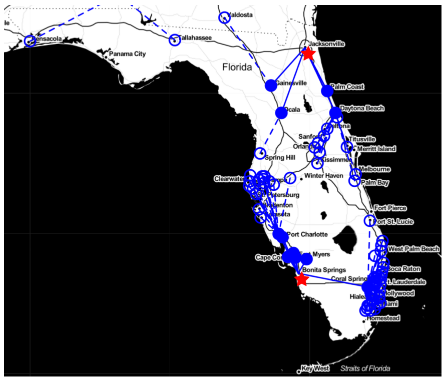


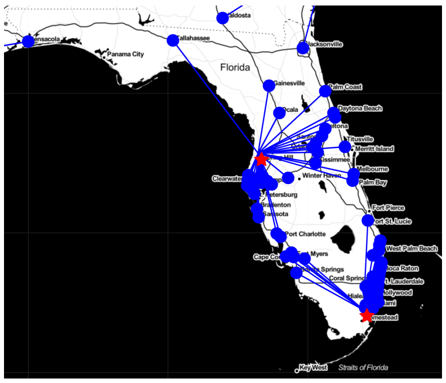
Figure 10: Optimal policies generated by the *exact* method for the South Florida area (left panel) and New England area (right panel).


(a) *Exact*


(b) *Algorithm 3-2*


(c) *Exact k-center*


(d) *Exact k-center + Alg. 3-1*


(e) *Approximate k-center*

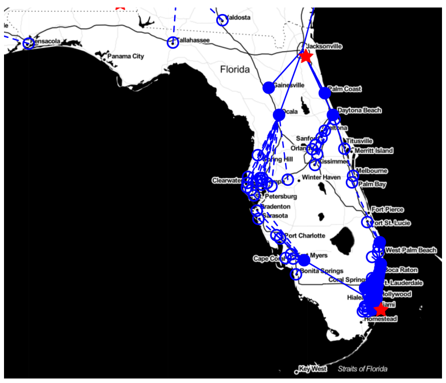
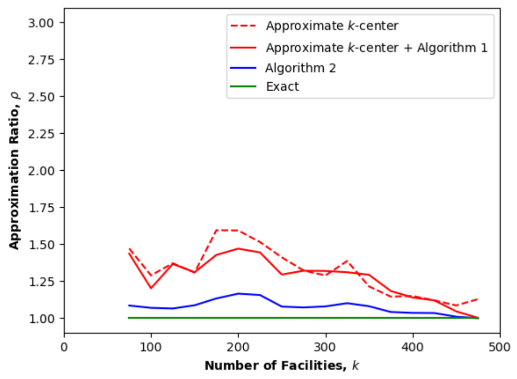
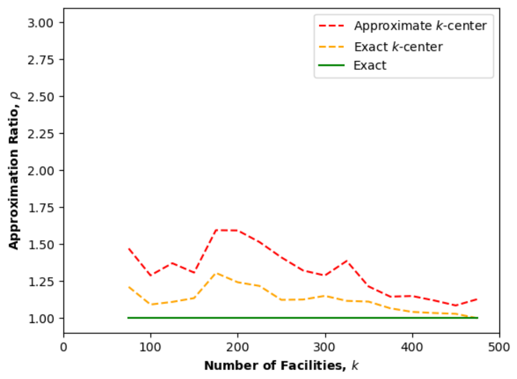

(f) *Approx. k-center + Alg. 3-1*

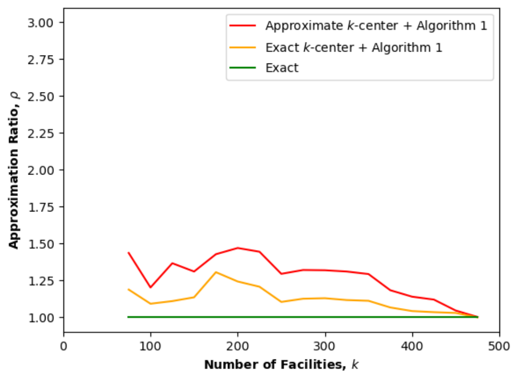
Figure 11: Florida portion of the global policy generated by each algorithm with $n = 1,000$ cities in the United States and $k = 50$ facilities.



(a) Approximation methods for the meet-in-the-middle k-center problem.



(b) Methods that do not consider the ability to meet in the middle.



(c) Methods that consider the ability to meet in the middle.

Figure 12: Approximation ratios, $\rho = z^{alg}/z^{*}$, for each method applied to the $n = 500$ most populous U.S. cities over a range over a range of 20 uniformly spaced values of k

3.8 Discussion

In this work we consider a variation of the k-center problem in which facilities can send agents to intermediate locations to meet-in-the-middle with clients. Our primary motivation for studying this problem stems from its application to vaccine distribution in geographically dispersed populations with limited transportation resources. To vaccinate parts of these populations, health workers take vaccines from medical facilities to intermediate locations, and patients travel to those intermediate locations to become vaccinated.

From a methodological standpoint, we studied the meet-in-the-middle k-center problem (3-2) under a variety of cost regimes; proposed a natural notion of distance between locations; developed a polynomial-time algorithm for solving subproblem (3-4) to determine optimal receiving and retrieving locations given a set of facility locations; presented two MILP formulations; and finally developed an approximation algorithm with performance guarantees for the related metric and equal metric cost regimes. Through computational experiments we observed that accounting for the ability to meet-in-the-middle can significantly reduce service distance, and we observed that our exact and approximation algorithms can solve significantly larger problem instances than the assignment MILP formulation (3-13).

There are a few interesting directions for future research. Naturally it is of interest to consider the meet-in-the-middle capability in the context of other facility location problems, such as those with fixed costs, supplies, demands, and capacities. Our meet-in-the-middle problem could be extended to consider the possibility of facilities delivering goods to a sequence (route) of locations from which nearby clients retrieve goods by traveling. We could consider different objectives, such as minimizing the total travel distance, or minimizing the demand-weighted travel distance. Finally, while we established that our analysis of Algorithm 3-2 is tight, we could explore whether other algorithms might provide better guarantees. Similarly, it would be interesting to explore algorithm performance under relaxed cost regimes, such as problems in which only Assumption 3.2.1 (i.e., metric costs) holds.

4.0 Recovering Locally Distinguishable Communities with Applications to Clustering Training Data

4.1 Introduction

Community detection (or *graph clustering*) involves clustering the vertices of a graph into vertex subsets called *communities*. A community is usually thought of as a subset of vertices that are more densely connected to each other than to vertices that are not in the subset. The amount that the communities are more densely internally than externally connected captures the extent to which the communities are *distinguishable* (or the strength of the community structure). Typically the goal in community detection problems is to either recover ground-truth communities or explore natural groupings of the vertices.

Data clustering problems are often recast as community detection problems through the notion of a *similarity graph*. A similarity graph is a graph whose vertices represent data points and whose edges indicate whether or not data points are "similar." (The edges may also have weights that capture the amount of similarity.) Recasting data clustering problems as community detection problems is particularly effective when it is of interest to find clusters of data points that are spread across space (e.g., when ground-truth communities are known to admit such structure).

In this work we consider a community detection problem in which the goal is to recover ground-truth communities of a vertex-attributed graph that are *locally distinguishable* in the sense that the ground-truth communities exhibit stronger community structure in more local parts of the graph (i.e., in subgraphs induced by vertices whose attributes are closer). A more precise (but still somewhat informal) description of the local distinguishability property that we consider is as follows.

The local distinguishability property. Communities are locally distinguishable if the following two conditions hold:

(i) The amount that the likelihood that there is an edge between two vertices in the same community exceeds the likelihood that there is an edge between two vertices in different communities (with the same attributes as the two vertices in the same community) increases as the distance (under some metric) between the attributes of the vertices decreases.

(ii) The likelihood that there is an edge between two vertices in the same ground-truth community (different ground-truth communities) increases (decreases) as the distance between the attributes of the vertices decreases.

Condition (i) alone captures the fact that locally distinguishable communities exhibit stronger community structure in more local parts of the graph. Condition (ii) is an additional condition that we enforce that is not necessarily implied by condition (i).

As we demonstrate in Subsection 4.2.1, the recovery problem of interest naturally arises when clustering training data that contains responses generated from different latent sources. More specifically, we show that the ground-truth communities (corresponding to the latent sources) of certain similarity graphs for the training data (whose edges capture whether or not the responses of training data points are similar) satisfy the local distinguishability property under mild conditions.

Locally distinguishable communities, however, may not be *globally distinguishable* (i.e., have higher internal than external connection density). Accordingly, the recovery problem poses a challenge for community detection algorithms that are designed based on standard notions of a community. In this work we design an algorithm specifically for the recovery problem. The algorithm first clusters the vertex set into *local vertex subsets*, then applies a standard (spectral) community detection algorithm to each graph induced by one of the local vertex subsets to obtain *local communities*, and finally agglomerates the local communities into global communities. We present a schematic of the algorithm along with a more detailed discussion about the algorithm in Subsection 4.2.2.

Our goal in this work is twofold. First, we aim to establish conditions under which our algorithm theoretically recovers when applied to the *local stochastic block model* (LSBM), a variation of the stochastic block model that incorporates the local distinguishability property. We direct the reader to Section 4.3 for a formal description of the LSBM and Subsection 4.3.1 for a summary of the main theoretical recovery guarantee that we develop for the algorithm. Second, we aim to evaluate the empirical performance of the method on LSBM instances and its viability for clustering real training data. To this end, we present a computational study in Section 4.8.

4.1.1 Organization

First in Section 4.2 we provide motivation for this work; specifically, we discuss the motivating application of clustering training data, present a schematic of the community detection algorithm that we propose, and discuss related work. Next we present a description of the LSBM in Section 4.3 along with the main recovery guarantee that we develop for the community detection algorithm. In Section 4.4 we introduce and study the algorithm that we will use to construct local vertex subsets. Next we present and analyze a spectral community detection algorithm for partitioning the local vertex subsets into local communities in Section 4.5. Then in Section 4.6 we introduce and analyze the algorithm that we will use to agglomerate the local communities. In Section 4.7 we present a full description of the community detection algorithm that we propose along with a formal recovery guarantee. Finally we present a computational study in Section 4.8, and we conclude with discussion and directions for future research in Section 4.10.

4.2 Motivation and Related Work

In Subsection 4.2.1 we discuss the motivating application of clustering training data. Next we provide a schematic of the community detection algorithm that we propose in Subsection 4.2.2. Finally in Subsection 4.2.3 we discuss related work.

4.2.1 Clustering Training Data

Consider training data $D = \{(x_i, y_i)\}_{i \in [n]} \subset \mathbb{R}^m \times \mathbb{R}$, where $[n] := \{1, \ldots, n\}$, that contains n feature-response pairs. Let $f_1, f_2 : \mathbb{R}^m \to \mathbb{R}$ be functions that we will refer to as *sources*. Suppose that each response is generated from one of the two sources: for each $i \in [n]$, either $y_i = f_1(x_i) + \epsilon_i$ or $y_i = f_2(x_i) + \epsilon_i$, where ϵ_i is random noise. (Later in this section we will place formal assumptions on the random noise.) Further suppose that the sources are latent, meaning the source of each response is unknown.

Example 4.2.1. As a running example, let us consider the cars dataset from the CMU Statlib library [69]. (We actually use the slightly revised version of the dataset from the UC Irvine Machine Learning Repository [66].) The dataset contains feature information (e.g., miles per gallon, car weight, and model year) for 406 different cars. Each car has a model year between 1970 and 1982. We restrict our attention to *newer* cars that have a model year in 1979-1982 and *older* cars that have a model year in 1970-1973. This reduces the dataset to $n = 243$ data points. Take x_i to be the weight of the i-th car (i.e., there is $m = 1$ feature), and take y_i to be miles per gallon (mpg) of the i-th car. Also take $f_1(x_i)$ to be the mpg of a newer car of weight x_i, and take $f_2(x_i)$ to be the mpg of an older car of weight x_i. (The sources are known in this case, but we will pretend that they are not.) See the first plot of Figure 13 for a scatter plot of the data. Observe that a newer car that has the same weight as an older car (naturally) tends to have higher mpg. The second plot in Figure 13 depicts overlaid smoothed estimates of the mean mpg (along with 95% confidence intervals) of newer and older cars as a function of their weight. We can think of these estimated functions as the sources f_1 and f_2, respectively. $\square$

Consider clustering the training data D; there are two key applications of this task. The first is training *clusterwise* models, and the second is labeling unlabeled data for training a classification model. We direct the reader to Subsection 4.2.3 for a thorough discussion of these applications and related work.

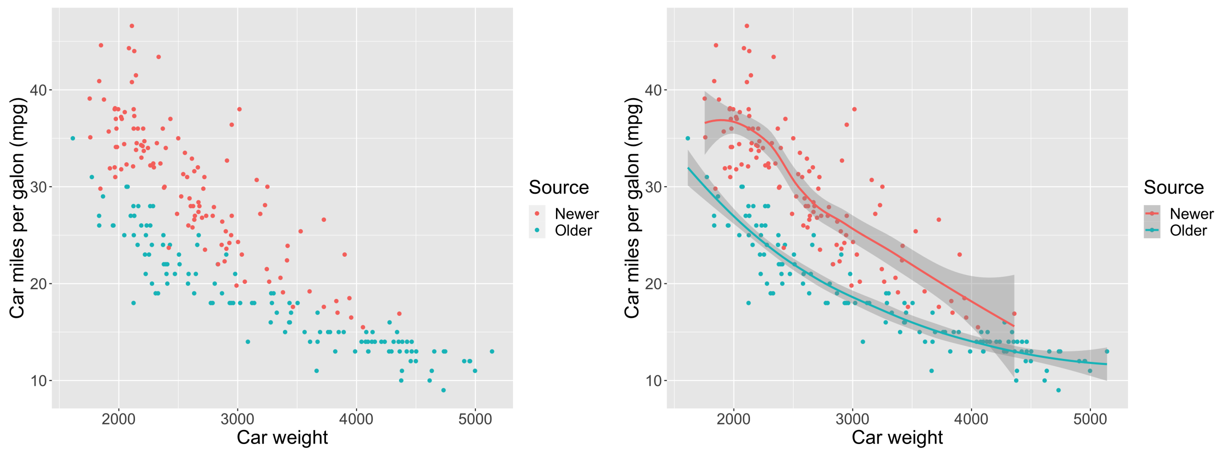


Figure 13: The car training data along with another plot of overlaid smoothed estimates of the mean mpg (along with 95% confidence intervals) of newer and older cars as a function of their weight.

Next we demonstrate that the ground-truth communities (corresponding to the sources) of certain similarity graphs for the training data are locally distinguishable under the following assumptions. The most strict assumption is that the sources are *separable*:

(i) For $\tau > 0$, it holds that $\inf_{x \in \mathbb{R}^n} f_1(x) - f_2(x) \geq \tau$. We say that the sources are *τ-separable*. In words, the first source outputs higher responses than the second source. Note that this condition is satisfied in Example 4.2.1.

The other assumptions are rather mild in comparison; we assume that the sources are *Lipshitz* and that the random noise terms are independent and identically distributed mean 0 normal random variables:

(ii) For $L > 0$, the sources are *L-Lipshitz*. That is, $|f_1(x_1) - f_1(x_2)| \leq L\|x_1 - x_2\|_2$ and $|f_2(x_1) - f_2(x_2)| \leq L\|x_1 - x_2\|_2$ for all $x_1, x_2 \in \mathbb{R}^n$.

(iii) The noise terms ϵ_i, $i \in [n]$ are independent and identically distributed mean 0 normal random variables.

A similarity graph G for D. We introduce the following vertex-attributed similarity graph G for the training data D. The vertex set of G is $V = [n]$, the attributes of vertex $i \in V$ are x_i (the features for the i-th data point), and there is an edge between vertices $i \neq j \in [n]$ if $|y_i - y_j| \leq \delta$, where δ is a parameter. (We discuss how to choose/tune δ below.) That is, there is an edge between vertices if the responses in the data points that they represent are close. We say that G is the *δ-similarity graph for D*. We also write $G = (V, E, x)$, where E is the edge set of G, and the columns of $x \in \mathbb{R}^{m \times n}$ are the vertex attributes (i.e., the i-th column of x is x_i).

Let $\delta > 0$, $G = (V, E, x)$ be the δ-similarity graph for D, and $i \neq j \in V$. Define the function $\phi_\delta : \mathbb{R} \to [0, 1]$ by $\phi_\delta(\gamma) := \mathbb{P}(|\epsilon_i - \epsilon_j + \gamma| \leq \delta)$ for $\gamma \in \mathbb{R}$. Note that ϕ_δ is decreasing on $\mathbb{R}_{\geq 0}$ because the random variable $\epsilon_i - \epsilon_j$ follows a mean 0 normal distribution. Also let $p_{ij} := \mathbb{P}(|y_i - y_j| \leq \delta)$ denote the probability (with respect to ϵ_i and ϵ_j) that there is an edge between vertices i and j.

Consider Proposition 4.2.1 below, which lower bounds (upper bounds) p_{ij} when the vertices i and j are in the same (different) ground-truth communities. The proposition readily follows from the three assumptions above.

Proposition 4.2.1. *Suppose that assumptions (i)-(iii) above hold. Let $\delta > 0$, G be the δ-similarity graph for D, and $i \neq j \in V$. If vertices i and j are in the same ground-truth community,*

$$p_{ij} \geq \phi_\delta(L\|x_i - x_j\|_2). \tag{4-1}$$

Otherwise, if the vertices are in different ground-truth communities,

$$p_{ij} \leq \phi_\delta(\tau - L\|x_i - x_j\|_2). \tag{4-2}$$

Proof. Suppose that vertices i, j are in the same ground-truth community. Furthermore, without loss of generality, suppose that $y_i = f_1(x_i) + \epsilon_i$ and $y_j = f_1(x_j) + \epsilon_j$. Then

$$\begin{aligned} p_{ij} &= \mathbb{P}(|y_i - y_j| \leq \delta) \\ &= \mathbb{P}(|\epsilon_i - \epsilon_j + f_1(x_i) - f_1(x_j)| \leq \delta) \\ &= \mathbb{P}(|\epsilon_i - \epsilon_j + |f_1(x_i) - f_1(x_j)|| \leq \delta) \\ &= \phi_\delta(|f_1(x_i) - f_1(x_j)|) \\ &\geq \phi_\delta(L\|x_i - x_j\|_2), \end{aligned}$$

the third equality follows from the fact that $\epsilon_i - \epsilon_j$ follows a mean 0 normal distribution, the third equality from the definition of ϕ_δ, and the inequality from the fact that f_1 is L-Lipschitz and ϕ_δ is decreasing.

Now suppose that vertices i, j are in different ground-truth communities. Furthermore, without loss of generality, suppose that $y_i = f_1(x_i) + \epsilon_i$ and $y_j = f_2(x_j) + \epsilon_j$. From a similar argument as above,

$$\begin{aligned} p_{ij} &= \phi_\delta(|f_1(x_i) - f_2(x_j)|) \\ &= \phi_\delta(|f_1(x_i) - f_2(x_i) + f_2(x_i) - f_2(x_j)|) \\ &\leq \phi_\delta(|f_1(x_i) - f_2(x_i)| - |f_2(x_i) - f_2(x_j)|) \\ &\leq \phi_\delta(\tau - L\|x_i - x_j\|_2) \end{aligned}$$

where the first inequality follows from the reverse triangle inequality and the fact that ϕ_δ is decreasing, and the second inequality follows from the fact the sources are τ-separable, L-Lipschitz, and ϕ_δ is decreasing. □

We are now setup to verify the two local distinguishability conditions (see Section 4.1); first we verify the second condition. Recalling that ϕ_δ is decreasing, inequality (4-1) tells us that the probability that there is an edge between vertices i in j in the same ground-truth community is larger (or at least guaranteed to be) if the distance $\|x_i - x_j\|_2$ between their attributes is smaller. Similarly, assuming that $\|x_i - x_j\|_2 \leq \tau/L$, inequality (4-2) tells us that the probability that there is an edge between vertices i in j in the different ground-truth communities is smaller (or at least guaranteed to be) if the distance $\|x_i - x_j\|_2$ between the attributes is smaller. Thus the second local distinguishability condition holds (assuming that the attributes are sufficiently close). Now let us verify the first local distinguishability condition. Suppose that vertices i and j are in the same ground-truth community, vertices $u \neq v \in V$ are in different ground-truth communities, and $\|x_i - x_j\|_2 = \|x_u - x_v\|_2$. Further suppose that $\|x_i - x_j\|_2 \leq \tau/(2L)$. From (4-1) and (4-2),

$$p_{ij} - p_{uv} \geq \phi_\delta(L\|x_i - x_j\|_2) - \phi_\delta(\tau - L\|x_i - x_j\|_2) \geq \phi(\tau/2) - \phi(\tau/2) = 0.$$

Furthermore, the function $h : [0, \tau/(2L)] \to \mathbb{R}_{\geq 0}$ defined by $h(r) = \phi_\delta(Lr) - \phi_\delta(\tau - Lr)$ is decreasing because its derivative $h'(r) = L(\phi'_\delta(Lr) + \phi'_\delta(\tau - Lr))$ is negative (as ϕ_δ is decreasing). It follows that $p_{ij} - p_{uv}$ is larger (or at least guaranteed to be) if the distance between the attributes $\|x_u - x_v\|_2 = \|x_i - x_j\|_2$ is smaller. Thus, the first local distinguishability property holds (assuming that the attributes are sufficiently close).

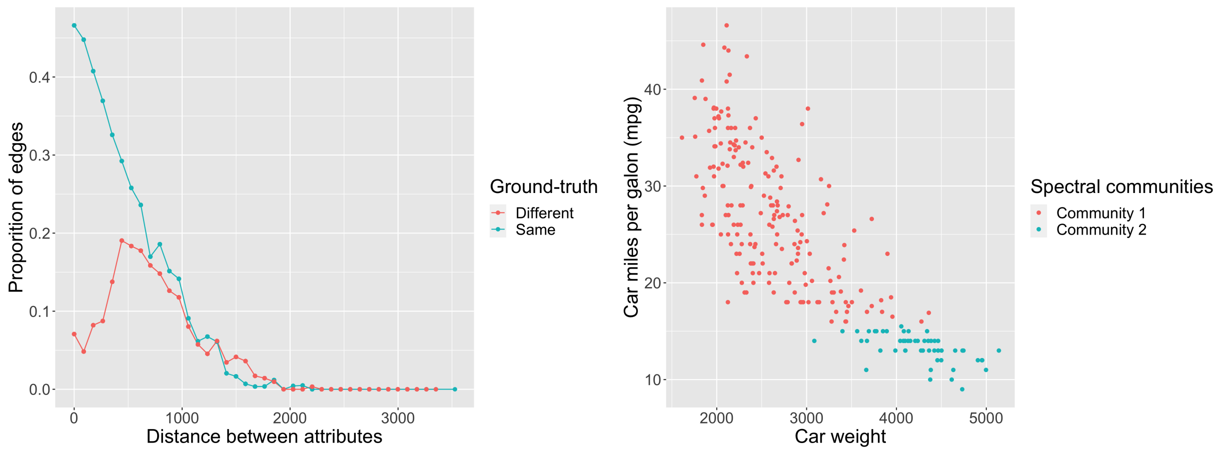


Figure 14: The proportion of edges between vertices in the same ground-truth community and different ground truth communities of the 2-similarity graph at different distances between attributes (car weight in this case), and the communities output by a standard spectral method applied to the adjacency matrix of the 2-similarity graph.

In light of this discussion, we revisit Example 4.2.1:

Example 4.2.2. Let us consider the 2-similarity graph for the training car data. We report the proportion of edges between vertices in the same ground-truth community and different ground truth communities at different distances between the attributes (car weight in this case) in the first plot of Figure 14. Observe that both local distinguishability conditions hold as long as the distance between attributes is sufficiently small (specifically, the difference between car weights is less than 500), resonating with the above argument.

Now consider applying the standard spectral community detection method (that uses the signs of the entries of eigenvector for the second largest eigenvalue of the adjacency matrix) to the 2-similarity graph. We report the communities that the method outputs in the second plot of Figure 14. Notice that the communities do not agree much with the ground-truth communities; this is not surprising in light of the fact that locally distinguishable ground-truth communities are not necessarily globally distinguishable.

Tuning the parameter δ in a semi-supervised setup. While our argument for local distinguishability does not rely on a particular specification of the parameter δ, the specification could impact the performance of a community detection algorithm applied to the δ-similarity graph. One option is to choose δ to produce the most distinguishable *local* communities under some pre-defined measure (e.g., the second largest eigenvalue). Another option is to use labels $\{z_i\}_{i \in I} \subseteq \{-1, +1\}$ for a subset of the data points (index by $I \subseteq [n]$) if such labels are available. We explore this direction in our computational experiments in Section 4.8, and we discuss connections with the semi-supervised learning literature in Subsection 4.2.3.

We conclude by noting that we will primarily focus on the community detection problem of interest for the remainder of this work, for the most part forgetting the underlying motivating application of clustering training data. An exception, however, is the computational study in Section 4.8, in which we use the community detection algorithm that we develop to cluster similarity graphs for training data that we introduce here.

4.2.2 Schematic of Community Detection Algorithm

We present a schematic of our community detection algorithm in Algorithm 4-1. Note that the algorithm takes a metric $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_{\geq 0}$ as input. As we discuss in more detail below, the metric will be used to cluster the attributes to construct local vertex subsets. Also note that the algorithm takes parameters (k, r) as input. We discuss these parameters in more detail below as well, along with Steps 1-3 of the algorithm. We present a more formal description of the steps as separate algorithms in Section 4.4, 4.5, and 4.6, respectively, and we present a more formal description of the community detection algorithm as a whole in Section 4.7.

Algorithm 4-1 Schematic of community detection algorithm.

```
Input: Vertex-attributed graph $G = (V, E, x)$, metric $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_{\geq 0}$, and parameters
       $(k, r)$
1: Construct $k$ local vertex subsets $V^{(i)} \subseteq V$, $i \in [k]$
2: Partition each local vertex subset $V^{(i)}$ into local communities $C_1^{(i)}$ and $C_2^{(i)} = V^{(i)} \setminus C_1^{(i)}$
3: Agglomerate the local communities into global communities $C_1$ and $C_2 = V \setminus C_1$
4: Return the global communities $C_1$ and $C_2$
```

Step 1: Local vertex subset construction. A *local vertex subset with respect to* x is a subset $V' \subseteq V$ of the form $V' = \{i \in V : d(x_i, c) \leq r\}$ for some $c \in X$ and $r > 0$. We refer to c and r as the *center* and *local radius* of V', respectively.

Step 1 of Algorithm 4-1 constructs k local vertex subsets $V^{(i)} \subseteq V$, $i \in [k]$ with respect to the input vertex attributes x. The number of local vertex subsets k is a parameter specified by the user. We will establish values of k that ensure theoretical recovery performance and discuss how to tune k in practice. To construct the local vertex subsets, we apply a k-center clustering algorithm to the vertex attributes; see Algorithm 4-2 in Section 4.4. The algorithm computes k centers c_i, $i \in [k]$ and then uses these centers to construct the local vertex subsets

$$V^{(i)} := \{j \in V : d(x_j, c_i) \leq r\}, \quad i \in [k],$$

where the local radius r is another parameter specified by the user. Like k, we will establish values of r that ensure theoretical recovery performance and discuss how to tune r in practice. We note that r must be chosen large enough that each vertex is in at least one of the local vertex subsets. Also, some vertices might belong to multiple local vertex subsets (so $V^{(i)}, i \in [k]$ will not necessarily partition V).

Step 2: Local community detection. Step 2 of Algorithm 4-1 partitions each local vertex subset $V^{(i)}$ into *local communities* $C_1^{(i)}$ and $C_2^{(i)} = V^{(i)} \setminus C_1^{(i)}$, which should be thought of as estimates of the *local ground-truth communities* (i.e., the intersection of the ground-truth communities and $V^{(i)}$). To obtain $C_1^{(i)}$ and $C_2^{(i)}$, we apply a spectral algorithm to (a recentered version of) the adjacency matrix $A^{(i)}$ of the subgraph $G[V^{(i)}]$ in G induced by $V^{(i)}$; see Algorithm 4-3 in Section 4.5. That is, $A^{(i)}$ is the principal submatrix of the adjacency matrix A of G that contains the columns and rows of A indexed by $V^{(i)}$. We will refer to $A^{(i)}$ as the *local adjacency matrix* of $V^{(i)}$.

We remark that existing analysis for spectral methods applied to the SBM does not readily lend itself to the setting of interest. Existing analysis relies on closed-form formula for the eigenvectors and eigenvalues of expected value of the adjacency matrix of the SBM that do not hold in the setting of interest. To circumvent this, we will establish spectral properties of the expected value of local adjacency matrices.

Step 3: Agglomeration of local communities. Step 3 of Algorithm 4-1 agglomerates the local communities into global communities C_1 and C_2. For each $i \in [k]$, define the local community pair $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$. In each iteration of our agglomeration method (Algorithm 4-4 in Section 4.6), we agglomerate two community pairs together using *vertex agglomeration*. The first iteration of the vertex agglomeration method proceeds as follows (the remaining iterations proceed similarly). We select the indices $i \neq j \in [k]$ that maximize $|V^{(i)} \cap V^{(j)}|$, and then we agglomerate $C^{(i)}$ and $C^{(j)}$ based on how many vertices are shared between these local community pairs.

4.2.3 Related Work

Below we discuss three related bodies of research.

The SBM and community detection. Naturally it is of interest to understand conditions under which community detection algorithms can recover ground-truth communities. Towards this end, a number of works study the theoretical performance of community detection algorithms applied to random graphs models that have built-in community structure. Most of these works consider the SBM (or some variation of it). In the SBM, n vertices are first randomly partitioned into ground-truth communities, and then edges are placed between pairs of vertices in the same community with probability p (the *within probability*) and between pairs of vertices in different communities with probability q (the *between probability*). Ground-truth communities are more distinguishable (or there is stronger community structure) if the difference $p - q$ is larger.

Classic work on the topic establishes lower bounds on the difference $p - q$ that guarantee that an algorithm of interest (numerous algorithms have been considered) *recovers* the ground-truth communities with high probability (i.e., with probability tending to 1 as $n \to \infty$) [9, 13, 16, 21, 22, 26, 74, 55, 83]; we discuss different notions of recovery in Section 4.3. While classic work lower bounds $p-q$, more recent work considers the case in which p and q are small as well, i.e. when the graph is sparse. On one hand, this is practical consideration, as graphs that arise in practice are often sparse. On the other hand, it is a theoretical consideration, as there is an information-theoretic phase transition in the sparse regime [2]. It is now known that semidefinite programming methods and the standard spectral method *exactly* recover (see Section 4.3) up to the information-theoretic threshold in the sparse regime [2, 3].

Our work is inherently different in two ways from these developments. First, in the LBSM (see Section 4.3), the within and between probability are not homogeneous (they are functions of the distance between vertex attributes), so the model is more challenging to study. Specifically, existing work on the SBM relies on analytic formulae that do not extend to our setting. Second, it is not clear that sparse graphs arise in the underlying application of labeling training data. In fact, if the points are densely distributed enough, then one could argue that the graphs are not sparse if the similarity parameter δ (see Subsection 4.2.1) is of the appropriate order. Based on these considerations, we establish a lower bound on a quantity that is analogous to the difference $p - q$ (see Section 4.3), rather than additionally considering a sparse regime of some sort.

We note that there are numerous studies (like ours) that consider vertex-attributed variations of the SBM [25, 30, 59, 90, 89, 88, 96]. To the best of our knowledge, the most closely related variation is the geometric block model [30]. In this model, there is an edge between two vertices if the their random attributes are close enough. The setup differs from ours in two ways. First, even if two vertices have the same attributes, it is not guaranteed that there will be an edge between them in our model. Second, we allow our algorithm to use the attributes, specifically a metric that captures the distance between them.

We conclude by directing the reader to the survey [1] for a comprehensive overview of existing work on the SBM.

Clustering training data. There are two key applications of clustering training data. The first application is training *clusterwise* models. When responses are generated from different sources, it might not be possible to adequately fit a model (especially an interpretable model, such as a linear regression model) to the data. To circumvent this, one could first cluster the training data and then fit a model to each cluster of data. The models could then be studied (in which case interpretability is important) or even aggregated for prediction purposes. Most of the work in this area has focused on *clusterwise linear regression*, in which the goal is to construct linear models for each cluster [24]. Our work is different in that it does not make any parametric modeling assumptions. A consequence of this, however, is that our methods are potentially only practical in applications that have low-dimensional feature spaces.

The second application is labeling unlabeled data for training a classification model. The goal in this application is to label the data points according to their source (i.e., cluster the training data) in order to construct a classification model that predicts the source of a new data point. The problem arises, for example, when constructing classification models for predicting whether or not a patient is taking their medication because labeled data (that captures whether or not a patient is taking their medication) is often not readily available or even obtainable [29, 43]. When partially labeled data is available, the problem is a semi-supervised learning problem; below we discuss work in this direction.

Transductive semi-supervised learning. The semi-supervised learning literature primarily considers semi-supervised binary classification problems; see the surveys [15, 78]. The data in these problems is comprised of features $\{x_i\}_{i\in[n]} \subset \mathbb{R}^m$ and labels $\{y_i\}_{i\in I} \subseteq \{-1, +1\}$ for a subset of the features (indexed by $I \subset [n]$). Recall from Subsection 4.2.1 that the semi-supervised version of clustering training data is a slightly different; specifically, we are given labels $\{z_i\}_{i\in I}$ for a subset of the feature-response data $\{(x_i, y_i)\}_{i\in[n]}$.

Methods for semi-supervised binary classification fall into one of two classes: *inductive* and *transductive*. Inductive methods use the data to directly construct a classification model. Transductive methods first use the data to label the unlabeled features and then use the resulting fully labeled data to construct a classification model. Transductive methods typically involve the following three steps. First they construct a similarity graph for the feature data. A common approach is to add an edge between two vertices if the corresponding features are close enough. After the graph is constructed, they create weights for the edges, using the labels for the subset of the points. Finally they apply a graph clustering method to the weighted graph to obtain a partition and hence labels.

In our approach, we place an edge between vertices if the corresponding responses are close enough. Also, we do not explicitly use the labeled data points (e.g., to construct edge weights), but we use them to tune the similarity parameter δ and the parameters (k, r) of Algorithm 4-1.

4.3 The Local Stochastic Block Model

Define $V := [n]$, where n is a positive integer. Let $\mathcal{U}$ denote the uniform distribution on $[0,1]^m$. Also let d_2 denote the Euclidean metric on $\mathbb{R}^m$ (i.e., $d_2(x_1, x_2) = \|x_1 - x_2\|_2$ for $x_1, x_2 \in \mathbb{R}^m$). Note that $\max_{x_1, x_2 \in X} d_2(x_1, x_2) \leq \sqrt{m}$. Finally let $p, q, \alpha, \beta \in [0,1]$.

The local stochastic block model (LSBM). A vertex-attributed graph $G = (V, E, x)$ together with ground-truth communities $C_1^* \subseteq V$, $C_2^* = V \backslash C_1^*$ *follow the LSBM under parameters* $(n, m, p, q, \alpha, \beta)$ (or we write $(G, C^*) \sim \text{LSBM}(n, m, p, q, \alpha, \beta)$, where $C^* = (C_1^*, C_2^*)$ is the pair of ground-truth communities) if the graph and communities are generated as follows.

(i) $C_1^* = \{i \in V : Z_i = 1\}$, where Z_i, $i \in V$ are independently drawn from the Bernoulli distribution with probability parameter $1/2$. That is, in SBM parlance, we consider a *symmetric* model.

(ii) The vertex attributes x_i, $i \in V$ are independently drawn from $\mathcal{U}$. The attributes x and ground-truth community assignments Z_i, $i \in V$ are independent.

(iii) For each $i \neq j \in V$, an edge is placed between vertices i and j (i.e., $\{i, j\} \in E$) with probability

$$p_{ij} = \begin{cases} p + \alpha\,(1 - d_2(x_i, x_j)/\sqrt{m}) & \text{if } i, j \in C_1^* \text{ or } i, j \in C_2^* \\ q - \beta\,(1 - d_2(x_i, x_j)/\sqrt{m}) & \text{otherwise,} \end{cases} \tag{4-3}$$

independently of the other distinct vertex pairs.

To ensure that (4-3) is well-defined, we assume that $p + \alpha \leq 1$ and $0 \leq q - \beta \leq 1$. We also assume that $p \geq q$, which guarantees that *within probability* $p + \alpha(1 - d_2(x_i, x_j)/\sqrt{m})$ is at least as large as the *between probability* $q - \beta(1 - d_2(x_i, x_j)/\sqrt{m})$:

$$p + \alpha(1 - d_2(x_i, x_j)/\sqrt{m}) \geq q - \beta(1 - d_2(x_i, x_j)/\sqrt{m}).$$

Note that if $\alpha = \beta = 0$, then the LSBM is exactly the (symmetric) SBM (with parameters p and q).

The local distinguishability property. Consider Figure 15, which presents plots of the probability p_{ij} as a function of scaled distance $d_2(x_i, x_j)/\sqrt{m}$ for different values of α and β. Assuming $\alpha > 0$, the within probability $p + \alpha(1 - d_2(x_i, x_j)/\sqrt{m})$ decreases as a function of the distance $d_2(x_i, x_j)$. That is, the probability that we add an edge between two vertices in the same ground-truth community decreases as the distance between their attributes grows. Now consider the between probability $q - \beta(1 - d_2(x_i, x_j)/\sqrt{m})$. If $\beta < 0$, then the between probability increases as a function of distance. That is, the probability that we incorrectly add an edge between two vertices in different ground-truth communities increases as distances increases. Lastly, we note that if $\alpha > -\beta$, then the difference of the within probabilities and between probabilities, namely $p - q + (\alpha + \beta)(1 - d_2(x_i, x_j)/\sqrt{m})$, decreases as a function of the distance $d_2(x_i, x_j)$.

Local and global difference. Define the *local difference*

$$D_\ell := p + \alpha - (q - \beta)$$

and the *global difference*

$$D_g := p - q.$$

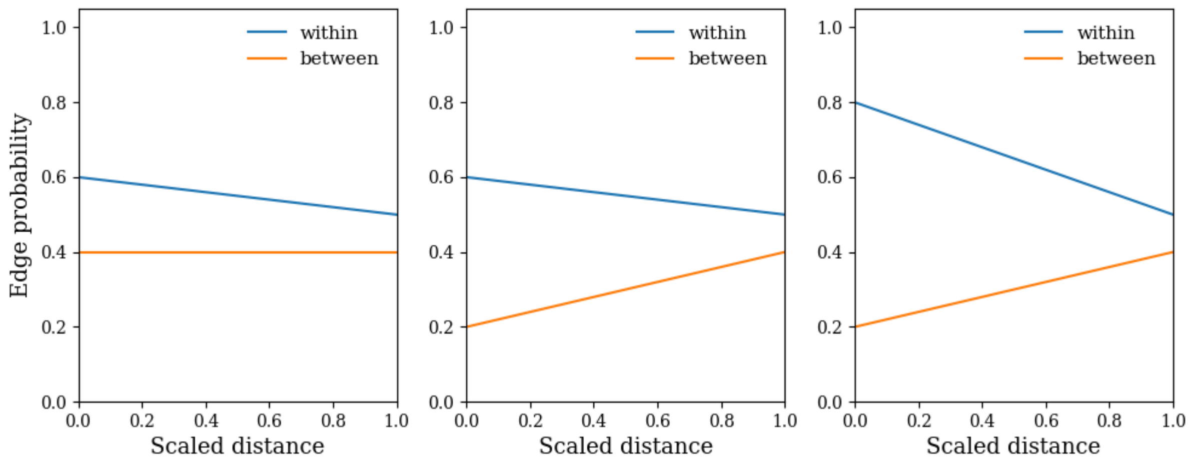


Figure 15: Edge probability p_{ij} as a function of scaled distance $d_2(x_i, x_j)/\sqrt{m}$ for different values of α and β. More specifically, $p = 0.5$ and $q = 0.4$ in all three plots; $(\alpha, \beta) = (0.1, 0), (0.1, 0.2), (0.3, 0.2)$ in the left, center, and right plots, respectively.

In the three panels of Figure 15, we have that $D_\ell = 0.2, 0.4, 0.6$ and $D_g = 0.1, 0.1, 0.1$, respectively. Note that $D_\ell \geq D_g$. The local and global difference capture how distinguishable ground-truth communities are at a local and global level. Analogous to results in the SBM literature, we will establish conditions on D_ℓ and D_g under which our algorithm recovers with high probability, bringing us to our next point.

Recovery in the LSBM. Let $\hat{C}_1, \hat{C}_2 \subseteq V$ such that $\hat{C}_1 \cap \hat{C}_2 = \emptyset$ and $\hat{C}_1 \cup \hat{C}_2 \neq \emptyset$. Similarly let $\tilde{C}_1, \tilde{C}_2 \subseteq V$ such that $\tilde{C}_1 \cap \tilde{C}_2 = \emptyset$ and $\tilde{C}_1 \cup \tilde{C}_2 \neq \emptyset$. Define the *agreement* between the pairs $\hat{C} = (\hat{C}_1, \hat{C}_2)$ and $\tilde{C} = (\tilde{C}_1, \tilde{C}_2)$ to be the proportion of the vertices in $\hat{C}_1 \cup \hat{C}_2 \cup \tilde{C}_1 \cup \tilde{C}_2$ that the pairs agree upon:

$$A(C, C^*) := \frac{\max\{|\hat{C}_1 \cap \tilde{C}_1| + |\hat{C}_2 \cap \tilde{C}_2|, |\hat{C}_1 \cap \tilde{C}_2| + |\hat{C}_2 \cap \tilde{C}_1|\}}{|\hat{C}_1 \cup \hat{C}_2 \cup \tilde{C}_1 \cup \tilde{C}_2|}.$$

(We define agreement in a slightly more general way than usual because we will examine the agreement of *local communities* in addition to *global communities*.)

Let $(G, C^*) \sim \mathrm{LSBM}(n, m, p, q, \alpha, \beta)$. A community detection algorithm that takes input $(G, d, p, q, \alpha, \beta)$ and outputs a global community pair $C = (C_1, C_2)$ (i.e., $C_1 \cup C_2 = V$) *exactly recovers* if

$$\mathbb{P}(A(C, C^*) = 1) = 1 - o(1)$$

and *almost exactly recovers* if

$$\mathbb{P}(A(C, C^*) = 1 - o(1)) = 1 - o(1).$$

Throughout our little-o notation is with respect to the number of vertices n. We treat m as a fixed constant, given the underlying motivation of low-dimensional feature/attribute spaces. In this work we will establish conditions on D_ℓ and D_g under which our algorithm almost exactly recovers.

We allow the algorithm to use these parameters (p, q, α, β) to obtain a theoretical guarantee; specifically, the algorithm uses the parameters in the local spectral community detection algorithm to recenter local adjacency matrices (see Section 4.5). In the computational experiments of (4.8), we will use an alternative spectral community detection method that does not require the parameters as input (but does not yield a theoretical guarantee, at least as far as we could show). Similarly, the algorithms that we develop run in the general setup of Subsection 4.2.2, in which the attributes are not necessarily in $[0, 1]^m$ and d is any metric, but we only establish theoretical guarantees for them when the attributes belong to $[0, 1]^m$ and $d = d_2$.

4.3.1 Almost Exact Recovery Result

First we set the stage for our results. The standard spectral method (that computes the top eigenvector of a centered version of the adjacency matrix) applied to the SBM almost exactly recovers if $p - q = \Omega(1/\sqrt{n})$, i.e. $(p - q)\sqrt{n} \to \infty$ as $n \to \infty$ [1]. The result follows from the Davis-Kahan theorem and a concentration bound for the operator norm; we will use similar techniques in our analysis of the local spectral community detection method in Section 4.5. While there are stronger guarantees for the standard spectral method (see the

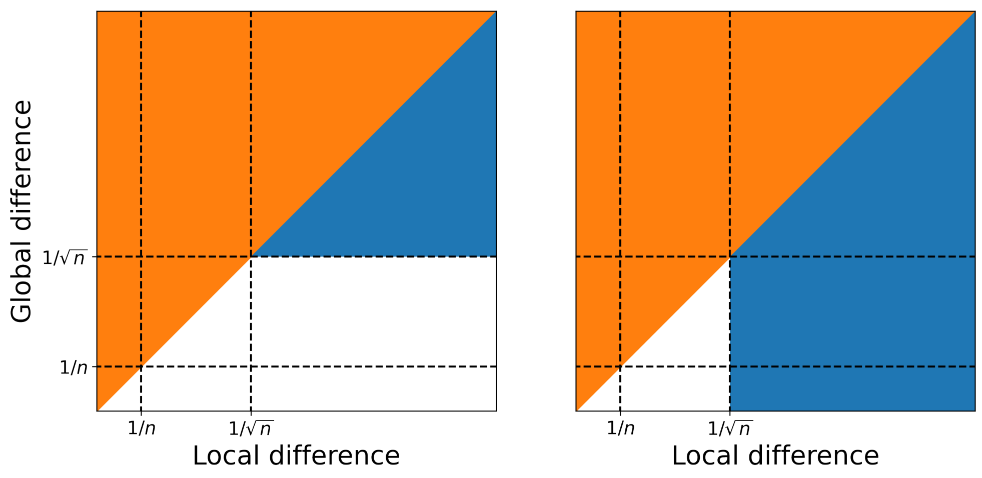


Figure 16: Comparison of almost exact recovery regions (depicted in blue) in terms of the local difference D_ℓ and global difference D_g that we would expect for a standard spectral method and that we establish for our community detection method. The orange area captures the region in which the global difference is larger than the local difference (a regime that we do not consider).

review of related work in Subsection 4.2.3), they require more involved arguments that we were unable to generalize to the setting of interest.

One could guess that applying the standard spectral method to the LSBM would almost exactly recover under the same condition that $D_g = p - q = \Omega(1/\sqrt{n})$ because $p - q$ lower bounds the difference of the within and between probability in the sense that $p - q \leq p - q + (\alpha + \beta)(1 - d_2(x_i, x_j)/\sqrt{m})$ for all $i \neq j \in V$. More strongly, it actually holds that $p - q \leq p_{ij} - p_{uv}$ for $i \neq j$ in the same ground-truth community and $u \neq v$ in different ground-truth communities. We show that we can remove the condition that $D_g = \Omega(1/\sqrt{n})$ by assuming that the local difference D_ℓ is sufficiently large, namely $D_\ell = \Omega(1/\sqrt{n})$:

Theorem 4.3.1. *If $D_\ell = \Omega(1/\sqrt{n})$, then Algorithm 4-5 (run with appropriate input) almost exactly recovers.*

Proof. See Section 4.7. □

In Figure 16 we present an illustrative comparison of the almost exact recovery regions (in terms of the local difference D_ℓ and global difference D_g) that we would expect for a standard spectral method and that we establish for our community detection method.

4.4 Constructing Local Vertex Subsets

We present the algorithm that we will use to construct local vertex subsets in Algorithm 4-2. Note that the algorithm takes vertex attributes x as input, as opposed to a vertex-attributed graph $G = (V, E, x)$; the algorithm will not use the edges of the graph to construct the local vertex subsets. Also note that the algorithm takes two input parameters: the number of local vertex subsets k and the local radius r for the output local vertex subsets.

Algorithm 4-2 Local vertex subset construction algorithm.

Input: Vertex attributes x, metric d, number of local vertex subsets k, and local radius r

```
1: Select any index $j \in [n]$
2: $U \leftarrow \{j\}$
3: while $|U| < k$ do
4:     $j \leftarrow \text{argmax}_{k \in [n] \setminus U} \min_{i \in U} d(x_i, x_k)$
5:     $U \leftarrow U \cup \{j\}$
6: end while
7: for $i \in [k]$ do
8:     Select any index $\ell \in U$
9:     $V^{(i)} \leftarrow \{j \in [n] : d(x_j, x_\ell) \leq r\}$
10:    $U \leftarrow U \setminus \{\ell\}$
11: end for
12: Return $V^{(i)}$, $i \in [k]$
```

Greedy k-center clustering. The first part of Algorithm 4-2 (specifically Steps 1-6) runs part of the greedy algorithm of [34] designed for the *k-center problem.* Given points (attributes) $x_1, \ldots, x_n \in X$ and a metric d on X, the goal in the k-center problem is to choose k of the n points to be centers to minimize the maximum distance (under d) from a point to its nearest center. In Step 1 the algorithm chooses any index $j \in [n]$; the index should be thought of as the index of the point x_j that will serve as the first "center." Next the algorithm initializes the set U of *center indices* in Step 2. Then in Step 4 the algorithm chooses the index of the next center to correspond to the point that is farthest from the centers whose indices have been chosen thus far. The algorithm adds the index to U in Step 5. The algorithm stops adding indices to U (i.e., the while loop terminates) once k indices have been chosen.

In Steps 7-11 the algorithm uses the center indices U to construct local vertex subsets $V^{(1)}, \dots, V^{(k)}$. Specifically, for each $\ell \in U$, the algorithm constructs a local vertex subset that contains the indices of all points that are within distance r to x_ℓ. In other words, the algorithm assigns each point to each center that is within distance r. Accordingly, a vertex could be either be in one local vertex subset, multiple local vertex subsets, or no vertex subsets. In contrast, the greedy algorithm of [34] assigns each point to a center that it is closest to, ensuring that each point is assigned to exactly one cluster.

Coverage, containment, and intersection properties. Our aim in running Algorithm 4-2 is to obtain local vertex subsets that (i) *cover* all of the vertices and (ii) individually *contain* enough vertices to recover local communities (later with a local community detection method), and (ii) *intersect* enough to eventually agglomerate via vertex agglomeration. Here we establish conditions under which these desired outcomes hold.

First we present a lower bound on k (in terms of r) that ensures Algorithm 4-2 returns local vertex subsets that cover all of the vertices. Our proof of Theorem 4.4.1 in Appendix 4.11 first constructs a $(r/2)$-net for $[0,1]^m$ (see Definition 4.11.1 in Appendix 4.11) of cardinality at most $(4\sqrt{m}/r)^m$ and then utilizes the fact that the greedy algorithm of [34] is a 2-approximation algorithm for the k-center problem.

Theorem 4.4.1. *Let $x \in [0,1]^m$. Further suppose that $0 < r < 1$ and $n \geq k \geq (4\sqrt{m}/r)^m$. Let $V^{(i)}$, $i \in [k]$ denote the local vertex subsets that Algorithm 4-2 outputs when run with input (x, d_2, k, r). Then the local vertex subsets $V^{(i)}$, $i \in [k]$ cover V; that is, $\cup_{i \in [k]} V^{(i)} = V$.*

Proof. See Appendix 4.11. □

Note that the lower bound on k (and hence n) in Theorem 4.4.1 grows exponentially with the dimension m of X. From a theoretical standpoint, this will not be an issue because we are investigating recovery performance as $n \to \infty$ and m is a fixed constant. From a practical standpoint, we are interested in the case in which the attribute space is low-dimensional.

The next result that we present (see Theorem 4.4.2 below) assumes that the attributes are independently drawn from the uniform distribution $\mathcal{U}$. The result depends on the quantity $\gamma(r) := \min_{x \in [0,1]^m} \mathbb{P}_{y \sim \mathcal{U}}(y \in B(x,r))$, where $r > 0$. The quantity $\gamma(r)$ lower bounds the probability that the attributes of a vertex in the LSBM are in a given ball of radius r centered at a point in $[0,1]^m$. Also, $\gamma(r) > 0$. It is possibly to lower bound by an exponentially small quantity in m, but to keep the presentation clean, we forgo doing this. The result also depends on the notion of an *intersection graph.* For $\eta > 0$, define the *η-intersection graph* of the local vertex subsets $V^{(i)}$, $i \in [k]$ to have vertex set $[k]$ and an edge between vertices $i, j \in [k]$ if $|V_i \cap V_j| \geq \eta$.

Theorem 4.4.2. *Suppose that $x_i \sim \mathcal{U}$, $i \in [n]$ are independent. Further suppose that $0 < r < 1$ and $n \geq k \geq (16\sqrt{m}/r)^m$. Let $V^{(i)}$, $i \in [k]$ denote the local vertex subsets that Algorithm 4-2 outputs when run with input (x, d_2, k, r). Then the $\frac{\gamma_{\mathcal{D}}(r/8)}{2}n$-intersection graph of the local vertex subsets is connected with probability at least*

$$1 - \left(16\sqrt{m}/r\right)^m \exp\left(-\gamma\left(r/8\right) n/8\right).$$

Proof. See Appendix 4.11. □

Theorem 4.4.2 shows that the $\frac{\gamma(r/8)}{2}n$-intersection graph of the local vertex subsets that Algorithm 4-2 outputs is connected with high probability. This implies that each local vertex subset contains a constant (in n) fraction of the vertices, namely $\frac{\gamma(r/8)}{2}$. Also we will see in Subsection 4.6 that we can successfully agglomerate local communities via vertex agglomeration if the local vertex subsets intersect on enough vertices precisely in the sense that the intersection graph is connected. The proof of Theorem 4.4.2 follows from a similar argument to the one that we use in the proof of Theorem 4.4.1 together with a Chernoff bound and the union bound.

4.5 Local Community Detection

In this section we propose and study the algorithm that we will use for local community detection. More specifically, let $G = (V, E, x)$ be a vertex-attributed graph that has ground truth-community pair $C^* = (C_1^*, C_2^*)$. Also let $V' \subseteq V$ be a local vertex subset with respect to x. We consider applying a spectral community detection algorithm to the local adjacency matrix A' of V' (i.e., the adjacency matrix of the subgraph $G[V']$ in G induced by V') to recover local ground-truth communities $C_1^* \cap V'$ and $C_2^* \cap V'$. We remark that the algorithm is specifically designed for the case when (G, C^*) is generated according to the LSBM. At the end of this section we describe an alternative local community detection method that we use in the computational study of Section 4.8.

Define $n' := |V'|$ be the number of vertices in V'. For the sake of exposition, we will assume throughout this section that $V' = [n']$. Suppose that r is the local radius of V'. Also for the sake of exposition, we define the *local multiplier* of V' as $\lambda := \frac{r}{\sqrt{m}}$ (i.e., the radius, rescaled), and we say V' is a λ-local vertex subset.

Algorithm 4-3 Local spectral community detection algorithm.

```
Input: Local adjacency matrix $A'$ of $V' = [n']$, local multiplier $\lambda$ of $V'$, and model parameters $(p, q, \alpha, \beta)$

1: Compute a top eigenvector $u_1$ of
       $$A' + (p + \alpha(1-\lambda))I - \frac{p + q + (\alpha - \beta)(1-\lambda)}{2} ee^\top$$
2: Return the local communities $C = (C_1, C_2)$ defined by
       $$C_1 = \{i \in [n'] : (u_1)_i < 0\} \quad \text{and} \quad C_2 = \{i \in [n'] : (u_1)_i \geq 0\}$$
```

We present a description of the local spectral community detection algorithm in Algorithm 4-3. In Step 1 the algorithm computes a top eigenvector of the *centered* local adjacency matrix

$$\tilde{A} := A' + (p + \alpha(1-\lambda))I - \frac{p + q + (\alpha - \beta)(1-\lambda)}{2} ee^\top, \tag{4-4}$$

where I is the $n' \times n'$ identity matrix, and e is the n'-dimensional vector of all ones. In Step

2 the algorithm uses the sign of the entries in u_1 to construct local communities C_1 and C_2. Some motivation for the algorithm is as follows.

Motivation. Let us continue to treat x and C^* as fixed, but suppose that A' is generated according to (4-3) under x and C^*, i.e.,

$$\mathbb{P}(A'_{ij} = 1) = \begin{cases} p_{ij}(x, C^*) & i \neq j \\ 0 & i = j \end{cases} \tag{4-5}$$

for $i \leq j \in [n']$, and $A_{ij} = A_{ji}$ for $i > j$. Define the $n' \times n'$ matrix $P' := \mathbb{E}[A']$, where the expectation is with respect to (4-5) (i.e., not additionally with respect to x or C^*, as we have assumed that they are fixed). Note that $P'_{ij} = p_{ij}(x, C^*)$ if $i \neq j$, and $P'_{ij} = 0$ if $i = j$. Also define $u \in \{-1, +1\}^{n'}$ by

$$u_i := \begin{cases} +1 & i \in C_1^* \\ -1 & i \in C_2^*, \end{cases},$$

and define the $n' \times n'$ matrix D by

$$D_{ij} := \begin{cases} \alpha\left(\lambda - \frac{d_2(x_i, x_j)}{\sqrt{m}}\right) & i \neq j \text{ and either } i, j \in C_1^* \text{ or } i, j \in C_2^* \\ -\beta\left(\lambda - \frac{d_2(x_i, x_j)}{\sqrt{m}}\right) & i \neq j \text{ and either } i \in C_1^*,\ j \in C_2^* \text{ or } i \in C_2^*,\ j \in C_1^* \\ 0 & i = j. \end{cases}$$

Note that, since V' is a λ-local vertex subset with respect to x, we have $D_{ij} \geq 0$ if vertices i, j are in the same ground-truth community, and $D_{ij} \leq 0$ if vertices i, j are in different ground-truth communities.

From (4-3), for $i \neq j$, we can write

$$p_{ij}(x, C^*) = \begin{cases} p + \alpha(1 - \lambda) + \alpha\left(\lambda - \frac{d_2(x_i, x_j)}{\sqrt{m}}\right) & i, j \in C_1^* \text{ or } i, j \in C_2^* \\ q - \beta(1 - \lambda) - \beta\left(\lambda - \frac{d_2(x_i, x_j)}{\sqrt{m}}\right) & \text{otherwise,} \end{cases}$$

and hence

$$P' + (p + \alpha(1 - \lambda))I = \frac{p + q + (\alpha - \beta)(1 - \lambda)}{2} ee^\top + \frac{p - q + (\alpha + \beta)(1 - \lambda)}{2} uu^\top + D.$$

It follows from (4-4) and $P' = \mathbb{E}[A']$ that

$$\mathbb{E}[\tilde{A}] = \left(\frac{p - q + (\alpha+\beta)(1-\lambda)}{2}\right) uu^\top + D. \tag{4-6}$$

As long as either $p - q > 0$ or $\alpha + \beta > 0$ and $\lambda < 1$ (ensuring that no entry of $\mathbb{E}[\tilde{A}]$ equals 0), we see from (4-6) that $\mathbb{E}[\tilde{A}]_{ij}$ is positive (negative) if the vertices i, j belong to the same (different) communities. It follows from the Perron-Frobenius theorem (applied to the positive matrix $\operatorname{diag}(u)\mathbb{E}[\tilde{A}]\operatorname{diag}(u)$) that the matrix $\mathbb{E}[\tilde{A}]$ has a unique top eigenvector, the signs of whose entries tell us the local ground-truth community assignments. Accordingly, by computing the top eigenvector of $\tilde{A}$ in Step 1 of Algorithm 4-3, we hope to recover local ground-truth community assignments.

Almost exact local recovery. Theorem 4.5.1 shows that Algorithm 4-3 almost exactly recovers local ground-truth communities if $D_\ell = \Omega\left(\frac{1}{\sqrt{n'}}\right)$ and $\lambda \leq \frac{1}{4}$ (i.e., if the radius of the local vertex subset is not too large). As discussed above, the condition that $p - q > 0$ or $\alpha + \beta > 0$ (and $\lambda < 1$, which is implied by $\lambda \leq \frac{1}{4}$) ensures that $\mathbb{E}[\tilde{A}]$ has all non-zero entries.

Theorem 4.5.1. *Suppose that $p - q > 0$ or $\alpha + \beta > 0$. Let $x \in \mathbb{R}^{n \times m}$, and let C^* be a global community pair. Suppose that $V' \subseteq V$ is a λ-local vertex subset with respect to x of cardinality $n' = |V'|$ such that $\lambda \leq \frac{1}{4}$. Further suppose that $A' \in \mathbb{R}^{n' \times n'}$ is generated according to* (4-5) *under x and C^*. Let C denote the output of Algorithm 4-3 when run with input $(A', \lambda, p, q, \alpha, \beta)$. Then there exist absolute constants $c_1, c_2 > 0$ such that*

$$\mathbb{P}\left(A(C, C^*) \geq 1 - \frac{c_1}{D_\ell^2 n'}\right) \geq 1 - \exp(-c_2 n').$$

Proof. See Appendix 4.12. □

Practical implementation. Centering the local adjacency matrix in Algorithm 4-3 requires the model parameters (p, q, α, β). The standard spectral method that computes the second eigenvector of A' (instead of the top eigenvector of $\tilde{A}$) bypasses this centering step and does not require any model parameters. We were not able to establish a guarantee for the standard spectral method, but we use it anyways in our computational study in Section 4.8 instead of Algorithm 4-3. We leave it as an open question to establish a theoretical guarantee for the standard spectral method applied locally (and globally); see Section 4.10.

4.6 Local Community Agglomeration

Let $G = (V, E, x)$ be a vertex-attributed graph that has ground-truth community pair $C^* = (C_1^*, C_2^*)$. Also let $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, $i \in [k]$ be local community pairs, i.e. $C_1^{(i)}, C_2^{(i)} \subseteq V$ such that $C_1^{(i)} \cap C_2^{(i)} = \emptyset$ for each $i \in [k]$. Assume that each vertex in V belongs to at least one of the local communities. In this section we propose an algorithm for agglomerating the local community pairs into a global community pair $C = (C_1, C_2)$ to recover C^*, presuming that the local community pairs have high enough agreement with their corresponding local ground-truth community pairs.

First we establish preliminary definitions. For $i \in [k]$, define $V^{(i)} = C_1^{(i)} \cup C_2^{(i)}$ to be the corresponding local vertex subset for $C^{(i)}$. Also, for $i \neq j \in [k]$, define the local communities $C^{(i \setminus j)} = (C_1^{(i \setminus j)}, C_2^{(i \setminus j)})$ by $C_\ell^{(i \setminus j)} := C_\ell^{(i)} \setminus V^{(j)}$ for $\ell \in [2]$.

We present a description of the agglomeration method in Algorithm 4-4. At each iteration the algorithm agglomerates two local community pairs together with *vertex agglomeration*. More specifically, in Step 3 the algorithm chooses the indices i and j of the two community pairs that intersect on the most vertices. Then in Steps 4-6 the algorithm agglomerates the communities within those pairs that share the most vertices. Exactly $k - 1$ vertex agglomerations are needed to obtain global communities; indeed, the while loop in the algorithm terminates after $k - 1$ iterations (as it holds that $|U| = 1$ by this point). The algorithm returns a community pair C that is a global community pair, as long each vertex in V belongs to at least one of the input local communities (as we assumed).

From local to global agreement. One would hope that if the agreement of each of the local community pairs is sufficiently high, then the agreement of the global community pair that Algorithm 4-4 outputs will be high. Here we present a result in this direction; see Theorem 4.6.1. The result additionally assumes that the η-intersection graph (for some $\eta > 0$) of the corresponding local vertex subsets is connected (which recall from Section 4.4 we can guarantee holds with high probability for certain values of η).

Theorem 4.6.1. *Let $G = (V, E, x)$ be a vertex-attributed graph that has ground-truth community pair $C^* = (C_1^*, C_2^*)$. Also let $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, $i \in [k]$ be local community pairs.*

Algorithm 4-4 Vertex agglomeration algorithm.

Input: Vertex attributed graph G, local community pairs $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, $i \in [k]$

```
1: $U \leftarrow [k]$
2: while $|U| > 1$ do
3:     $(i, j) \leftarrow \text{argmax}_{i \neq j \in U} |V^{(i')} \cap V^{(j')}|$
4:     $(a, b) \leftarrow \text{argmax}_{(a',b') \in \{(1,2),(2,1)\}} |C_1^{(i)} \cap C_{a'}^{(j)}| + |C_2^{(i)} \cap C_{b'}^{(j)}|$
5:     $C_1^{(i)} \leftarrow C_1^{(i)} \cup C_a^{(j \setminus i)}$
6:     $C_2^{(i)} \leftarrow C_2^{(i)} \cup C_b^{(j \setminus i)}$
7:     $U \leftarrow U \setminus \{j\}$
8: end while
9: $C \leftarrow C^{(i)}$, where $i \in U$ is the only index left in $U$
10: Return community pair $C$
```

Suppose that each vertex in V belongs to at least one of the local communities. Also suppose that the η-intersection graph of $V^{(i)}$, $i \in [k]$ is connected. Finally suppose that

$$\min_{i \in [k]} A(C^{(i)}, C^*) \geq 1 - \frac{\zeta \eta}{2^{k+1} n},$$

where $0 \leq \zeta \leq 1$. Then Algorithm 4-4 applied with input $(G, (C^{(i)})_{i \in [k]})$ outputs a global community pair $C = (C_1, C_2)$ that satisfies

$$A(C, C^*) \geq 1 - \frac{\zeta \eta}{4n}.$$

Proof. See Appendix 4.13. □

Note that Theorem 4.6.1 requires that the disagreement of each of the local community pairs is exponentially small in k. It might be possible to improve this dependence on k in the analysis, but such a result is not required to establish almost exact recovery results with respect to n. We leave this as a direction for future research. More generally, it is of interest to consider other agglomeration methods as well (e.g., optimization-based agglomeration methods); see Section 4.10 for further discussion.

4.7 Community Detection Algorithm

We present a formal description of the community detection algorithm in Algorithm 4-5. Step 1 of the algorithm runs Algorithm 4-2 to obtain local vertex subsets $V^{(i)}$, $i \in [k]$. Next Steps 2-4 apply Algorithm 4-3 to each of the local vertex subsets to obtain local community pairs $C^{(i)}$, $i \in [k]$. Finally the algorithm agglomerates the local community pairs into a global community pair in Step 5 using Algorithm 4-4.

Algorithm 4-5 Community detection algorithm.

```
Input: Vertex-attributed graph $G = (V, E, x)$, metric $d$, algorithm parameters $(k, r)$, and
       model parameters $(p, q, \alpha, \beta)$
1: Compute local vertex subsets $V^{(i)}$, $i \in [k]$ using Algorithm 4-2 run with input $(x, d, k, r)$
2: for $i = 1, \ldots, k$ do
3:     Run Algorithm 4-3 with input $(A^{(i)}, \frac{r}{\sqrt{m}}, p, q, \alpha, \beta)$ to obtain local community pair
       $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, where $A^{(i)}$ is the local adjacency matrix $A^{(i)}$ of $V^{(i)}$
4: end for
5: Run Algorithm 4-4 with input $(G, (C^{(i)})_{i\in[k]})$ to obtain $C = (C_1, C_2)$
6: Return $C = (C_1, C_2)$
```

Almost exact recovery. Theorem 4.7.1 below presents conditions that ensure Algorithm 4-5 (under specific input specifications) almost exactly recovers LSBM communities. Our proof of the theorem follows from Theorem 4.4.1, 4.4.2, 4.5.1, and 4.6.1. Note that Theorem 4.3.1 follows from Theorem 4.7.1.

Theorem 4.7.1. *Suppose that $n \geq (64\sqrt{m})^m$. Let $(G, C^*) \sim \mathrm{LSBM}(n, m, p, q, \alpha, \beta)$. Also let C denote the global community that Algorithm 4-5 outputs when run with input G, d_2, $(k, r) = \left(\lceil (64\sqrt{m})^m \rceil, \frac{1}{4}\right)$, and (p, q, α, β). Then there are constants $c_i(m) > 0$, $i \in [4]$ (that depend on m) such that if*

$$D_\ell \geq \frac{c_1(m)}{\sqrt{n}}, \tag{4-7}$$

then it holds that

$$\mathbb{P}\left(A(C, C^*) \geq 1 - \frac{c_2(m)}{D_\ell^2 n}\right) \geq 1 - c_3(m)\exp(-c_4(m)n).$$

Proof. See Appendix 4.14. $\square$

The supposition that $n \geq (64\sqrt{m})^m$ in Theorem 4.7.1 is included to ensure that $k = \lceil (64\sqrt{m})^m \rceil \leq n$. Condition (4-7) is required to guarantee that local community agreement is high enough to achieve "successful" agglomeration.

Going into more detail, the proof of Theorem 4.7.1 reveals that $c_1(m)$ is doubly exponentially large in m. (This in turn through (4-7) implies an even larger lower bound on n than $(64\sqrt{m})^m$.) The intuition is as follows. Recall that Theorem 4.6.1 requires local disagreement to be exponentially small in k. Accordingly, because Theorem 4.7.1 requires k to exponentially large in m, we must set $c_1(m)$ to be doubly exponentially large in m.

4.8 Computational Experiments

Our first goal in this computational study is to evaluate the empirical recovery performance of Algorithm 4-5 applied to LSBM instances. Our second goal is to investigate its viability for clustering training data.

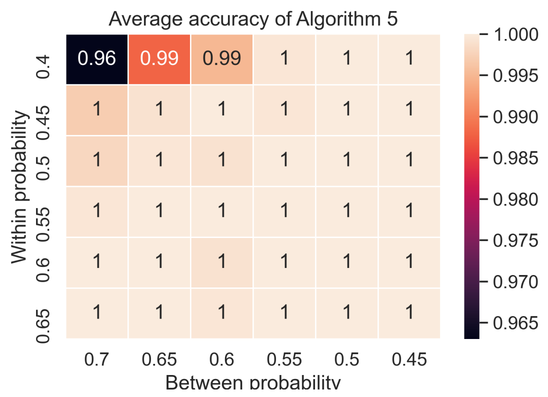
We consider the following modified implementation of Algorithm 4-5 that does require model parameters (p, q, α, β) as input. Instead of applying Algorithm 4-3 in Steps 2-4 to construct local communities, we apply the standard spectral method to the local adjacency matrices. In our experiments with clustering data, we tune the input parameters (k, r) to a subset of the data that is assumed to be labeled. Specifically, we consider $k \in \{2, \ldots, 10\}$ and $r \in \{0.1, 0.2, \ldots, 0.5\}$ such that $2rk \geq 1$. The condition $2rk \geq 1$ ensures that it is possible to cover $[0, 1]$ with k Euclidean balls of radius r (intervals of length $2r$). (If Algorithm 4.4 does not output local vertex subsets that cover V, we arbitrarily assign uncovered vertices to one community after Algorithm 4-5 terminates.) We also tune the similarity parameter δ. As a benchmark, we compare the performance of Algorithm 4-5 with the standard spectral method; we also tune the similarity parameter for the benchmark.

We consider recovering LSBM communities in Subsection 4.8.1, clustering synthetic training data from Lipschitz sources in Subsection 4.8.2, and clustering the car dataset from Subsection 4.2.1 in Subsection 4.8.3.

4.8.1 Recovering LSBM Communities

We generate $(G, C^*) \sim \text{LSBM}(n, 1, p, q, \alpha, \beta)$ with the following specification of parameters: $n = 200$, $p, q \in \{0.4, 0.45, \ldots, 0.7\}$, and $\alpha = \beta = .3$. Under each specification of the parameters p and q, it holds that $p+\alpha > q-\beta$ (i.e., the ground-truth communities are locally distinguishable). Under some specifications, $p > q$, $p = q$, or $p < q$ (i.e., the parameters are not globally distinguishable).

We generate 10 LSBM instances under each specification of parameters and apply Algorithm 4-5 along with the standard spectral method to each of the instances. When applying Algorithm 4-5 to each instance, we give it the benefit of the doubt and choose the parameter specification that yields the highest accuracy. (In subsequent sections we actually tune the parameters.) We report average accuracy of both algorithms across all of the instances in the heat plots of Figure 17.



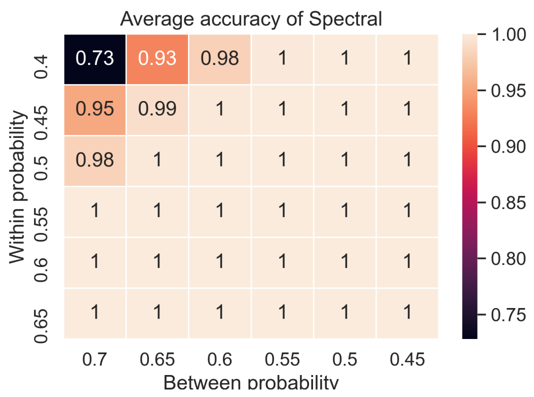


Figure 17: Average accuracy of Algorithm 4-5 and the standard spectral method against the within probability p and between probability q.

We observe that when $p \geq q$, the algorithms perform similarly, even when $p - q = 0$. This is interesting in light of the discussion in Subsection 4.3.1, i.e., perhaps the standard spectral algorithm almost exactly recovers when $D_\ell = \Omega(1/\sqrt{n})$ and $p - q \geq 0$. However, when $p \ll q$, we see that Algorithm 4-5 outperforms the standard spectral method. This is not surprising in light of the fact that the ground-truth communities are locally but not globally distinguishable in this regime.

4.8.2 Clustering Training Data from Separable, Lipschitz Sources

We revisit the setup from Subsection 4.2.1. Our goal is to understand how the Lipschitz constant L of the sources and the percentage of labeled data points (i.e, $|I|/n$) impacts the accuracy of Algorithm 4-5. (The percentage of labeled data points will impact our ability to tune the parameters (k, r) and δ.) We consider one-dimensional linear sources (i.e., $m = 1$) defined by $f_1(x) = 1/2 + Lx$ and $f_2(x) = -1/2 + Lx$ for $x \in \mathbb{R}$, where $L > 0$. The sources are separable and L-Lipschitz. We independently generate the noise terms ϵ_i, $i \in [n]$ from the normal distribution that has a mean of 0 and a standard deviation of 1/10.

To generate an instance, we first independently sample $n = 200$ features x_i, $i \in [200]$ from the uniform distribution on $[0, 1]$. Then we uniformly at random partition the features into two equally sized subsets (i.e., each subset contains 100 features). Next we a generate a response $y_i = f_j(x_i) + \epsilon_i$ for each feature x_i, where $j \in \{1, 2\}$ captures which subset of the partition that the feature belongs to. We use the responses to generate the δ-similarity graph for the training data $D = \{(x_i, y_i)\}_{i\in[200]}$. Finally, we uniformly at random label $|I|$ of the data points.

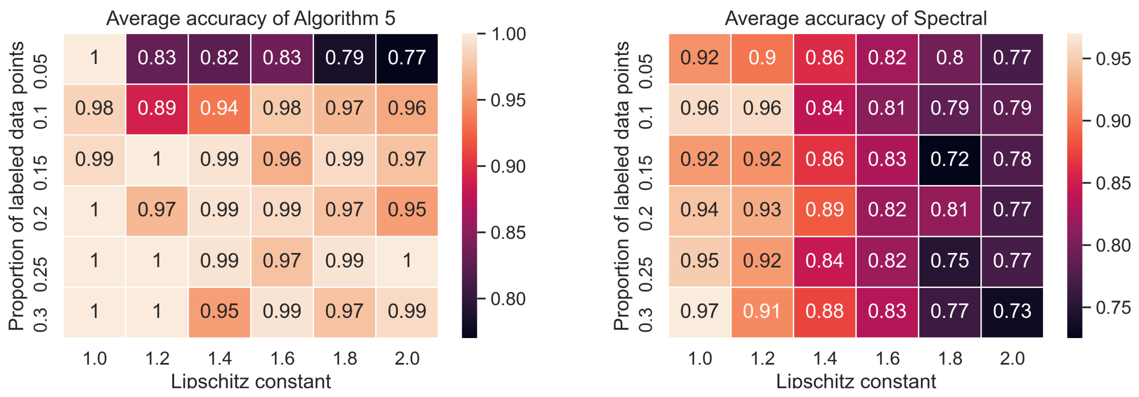


Figure 18: Average accuracy of Algorithm 4-5 and the standard spectral method against the Lipschitz constant L and the percentage $|I|/n$ of labeled data.

For each $L \in \{1, 1.2, \ldots, 2\}$ and $|I|/n \in \{.05, .1, \ldots, .3\}$, we generate 10 instances. We apply Algorithm 4-5 and the standard community detection method to each of these instances. More specifically, for each value of (k, r) (specified earlier at the beginning of

Section 4.8) and $\delta \in \{0.2, 0.4, \ldots, 1\}$, we apply Algorithm 4-5 and choose the output communities that give the highest accuracy on the subset of labeled data. Similarly, for each $\delta \in \{0.2, 0.4, \ldots, 1\}$, we apply the standard spectral method and choose the output communities that give the highest accuracy on the subset of labeled data. We report the average accuracy of both methods in the heat plots of Figure 18.

We observe in general that Algorithm 4-5 outperforms the standard spectral method, especially on instances that have a larger Lipschitz constant, as the performance of the standard spectral method naturally deteriorates as the the Lipschitz constant gets larger. While the performance of the spectral method does not change much with the amount of labeled data, the performance of Algorithm 4-5 significantly improves when at least 10% (versus 5%) of the data is labeled.

4.8.3 Clustering the Car Dataset

Recall the car dataset from Subsection 4.2.1. In this subsection we consider using Algorithm 4-5 to cluster the dataset under different amounts $|I|$ of labeled data.

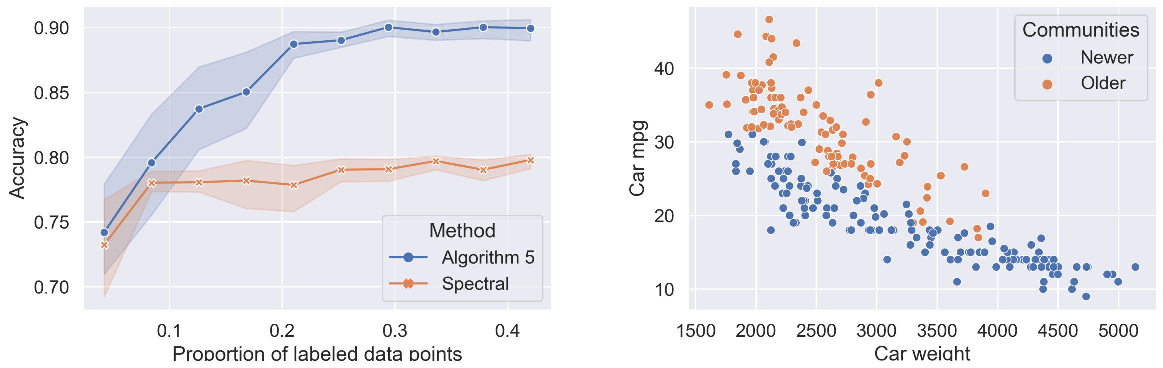


Figure 19: Average accuracy (with 95% confidence intervals) of Algorithm 4-5 and the standard spectral method against the percentage $|I|/n$ of labeled data, along with predicted communities output by Algorithm 4-5.

First we apply the transform $(x_i - \min_i x_i)/(\max_i x_i - \min_i x_i)$ to the features (i.e. the car weights) so that they take on values in $[0, 1]$. (This allows us to "successfully" tune the parameters (k, r) of Algorithm 4-5 over the values specified at the beginning of Section 4.8). Similar to Subsection 4-5, we uniformly at random label $|I|$ of the data points. For each value $|I| \in \{10, 20, \ldots, 100\}$, we generate 10 instances of labeled data in this way. We apply Algorithm 4-5 and the standard community detection method to each of the instances. We tune the parameters of the methods in the same way as Subsection 4.8.2, but we tune over $\delta \in \{1, \ldots, 10\}$ instead. We report average accuracy across the different fractions of available labeled data in Figure 19. We also present an illustration of communities output by Algorithm 4-5 that have an accuracy of approximately 0.91.

Both methods on average find communities that have higher agreement with the ground-truth communities when more data is available. Algorithm 4-5, however, tends to find communities with higher agreement, especially when more data is available. We observe in the second plot of Figure 19 Algorithm 4-5 fails to identify correct community structure amongst cars with very low or very high weight, as these cars typically only come from one source, either newer or older cars, respectively.

4.9 Computational Experiments with Edge Agglomeration

As a future research direction, we consider augmenting our community detection Algorithm 4-5 with *edge agglomeration*. Recall that the agglomeration step (i.e., Step 5 of Algorithm 4-5) relies on vertex agglomeration, which joins local communities that intersect on a sufficiently large number of vertices, more specifically, C'_{i1}, C'_{i2} and C'_{j1}, C'_{j2} when $|(C'_{i1} \cup C'_{i2}) \cap (C'_{j1} \cup C'_{j2})| \geq \eta$. For graphs with *strong local distinguishibility*, Algorithm 4-5 may terminate before all local communities have been agglomerated into global communities. That is, Algorithm 4-5 terminates when $|(C'_{i1} \cup C'_{i2}) \cap (C'_{j1} \cup C'_{j2})| > \eta$ for all unagglomerated communities C'_{i1}, C'_{i2} and C'_{j1}, C'_{j2}.

As an extreme but illustrative example, consider the hypothetical graph in Figure 20. Each vertex represents a politician who lives in either the city of Pittsburgh, Pennsylvania or the city of Boston, Massachusetts. The ground-truth communities correspond to the democrat (blue) and republican (red) political affiliations. Naturally it is more likely that there is an edge between vertices representing politicians living in the same city (regardless of whether the politicians have same political affiliation). If Algorithm 4-5 were to be applied using the local vertex subsets indicated as ellipses in Figure 20, the algorithm would terminate prior to joining the local communities into global communities (i.e., because $|(C'_{i1} \cup C'_{i2}) \cap (C'_{j1} \cup C'_{j2}) = \emptyset|$).

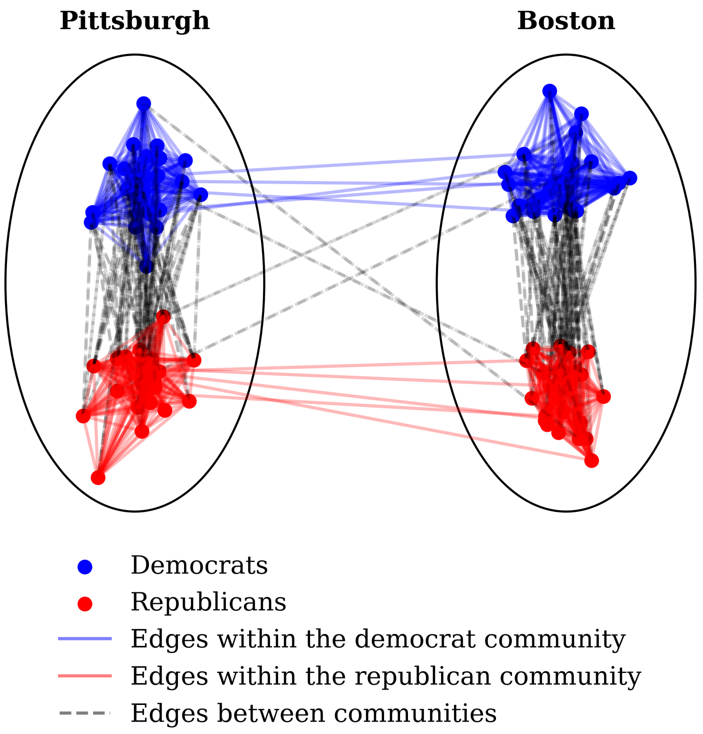

Pittsburgh
Boston
Democrats
Republicans
Edges within the democrat community
Edges within the republican community
Edges between communities

Figure 20: A hypothetical LBSN of democrat (blue) and republican (red) politicians who live in the cities of Pittsburgh, Pennsylvania and Boston, Massachusetts.

To prevent early termination of Algorithm 4-5, we consider an edge agglomeration step in which, at the i-th iteration, we agglomerate local community C'_{i1} with the current global community that it shares the most edges with, and we agglomerate the local community C'_{i2}

with the other current global community. More specifically,

$$\begin{aligned} &C_1' \leftarrow C_1' \cup (C_{i1}' \setminus C_2') \text{ and } C_2' \leftarrow C_2' \cup (C_{i2'} \setminus C_1') \text{ if } |E(C_1', C_{i1}')| \geq |E(C_2', C_{i1}')|, \\ &C_1' \leftarrow C_1' \cup (C_{i2}' \setminus C_2') \ \text{ and } C_2' \leftarrow C_2' \cup (C_{i1}' \setminus C_1') \text{ otherwise.} \end{aligned} \tag{4-8}$$

In Algorithm 4-6, we present a modified community detection algorithm that includes edge agglomeration in Step 6.

Algorithm 4-6 Modified community detection algorithm with edge agglomeration.

```
Input: Vertex-attributed graph $G = (V, E, x)$, metric $d$, model parameters $(p, q, \alpha, \beta)$, and
       algorithm parameters $(k, r)$

1: Compute local vertex subsets $V^{(i)}$, $i \in [k]$ using Algorithm 4-2 run with input $(x, d, k, r)$

2: for $i = 1, \dots, k$ do
3:     Run Algorithm 4-3 with input $(A^{(i)}, r/\sqrt{m}, p, q, \alpha, \beta)$ to obtain local community pair
       $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, where $A^{(i)}$ is the local adjacency matrix of $V^{(i)}$
4: end for
5: Agglomerate the local community pairs into pair $C = (C_1, C_2)$ using Algorithm 4-4 with
   input $(G, (C^{(i)})_{i \in [k]})$
6: If Step 5 completes prior to assigning all vertices to a global community such that
   $C_1 \cup C_2 = V$, then agglomerate the remaining local community pairs into $C = (C_1, C_2)$
   using edge agglomeration per (4-8)
7: Return $C = (C_1, C_2)$
```

In the following subsections, we study the performance of Algorithm 4-6 on a real-world social network without ground-truth communities, as well as to the task of image segmentation.

4.9.1 Experiment with an Empirical Social Network

We first consider a location-based online social network called Brightkite that consists of 58,228 vertices and 214,078 undirected edges ([49], online data accessed June 14, 2022). Each vertex represents a user (person), and an undirected edge exists between two vertices if there is a friendship between the corresponding users. The dataset also includes about 4.5 million user check-in locations, each specified by latitude and longitude. The Brightkite dataset is used by [20], along with others, to study user movement in location-based social networks.

We start by picking a random seed vertex in Washington, D.C. and and a random seed vertex in Philadelphia, PA. For each of these vertices, we perform a breadth-first search for connected vertices whose average check-in location is within a 50-mile radius of the seed, until two 125-vertex subgraphs are formed. The breadth first search ensures that each of the two subgraphs does not have disconnected components. Figure 21 depicts the resulting subgraphs, and we see that they are connected to each other by user friendships between the two cities. This graph is similar to an LSBM instance in the sense that edge likelihood changes as a function of distance between vertices.

While the Brightkite dataset does not identify a ground-truth community for each vertex, we can compare the partitions generated by Algorithm 4-6 and spectral partitioning. Figure 22 illustrates that the spectral algorithm (left) partitions the graph by city (Washington, D.C. and Philadelphia), whereas Algorithm 4-6 returns communities that are distinguishable within each city (i.e., Algorithm 4-6 returned locally distinguishable communities). Factors that might generate such locally distinguishable communities in social networks might include political affiliation, brand preferences, and media preferences.

4.9.2 Experiment with Images

We conclude our experiments by illustrating how Algorithm 4-6 can be applied to image segmentation. Image segmentation seeks to divide an image into distinct objects. The machine vision community has performed an enormous amount of research in this area since the 1970's, and one class of the resulting methods involves graph partitioning. Surveys of graph theoretic approaches to image segmentation are provided by [63] and [19]. The basic idea is to construct a graph in which each vertex corresponds to a pixel, and each edge is weighted by some measure of color similarity between two pixels. The resulting graphs can be huge, even for moderately resolute images, and the process is made more tractable by

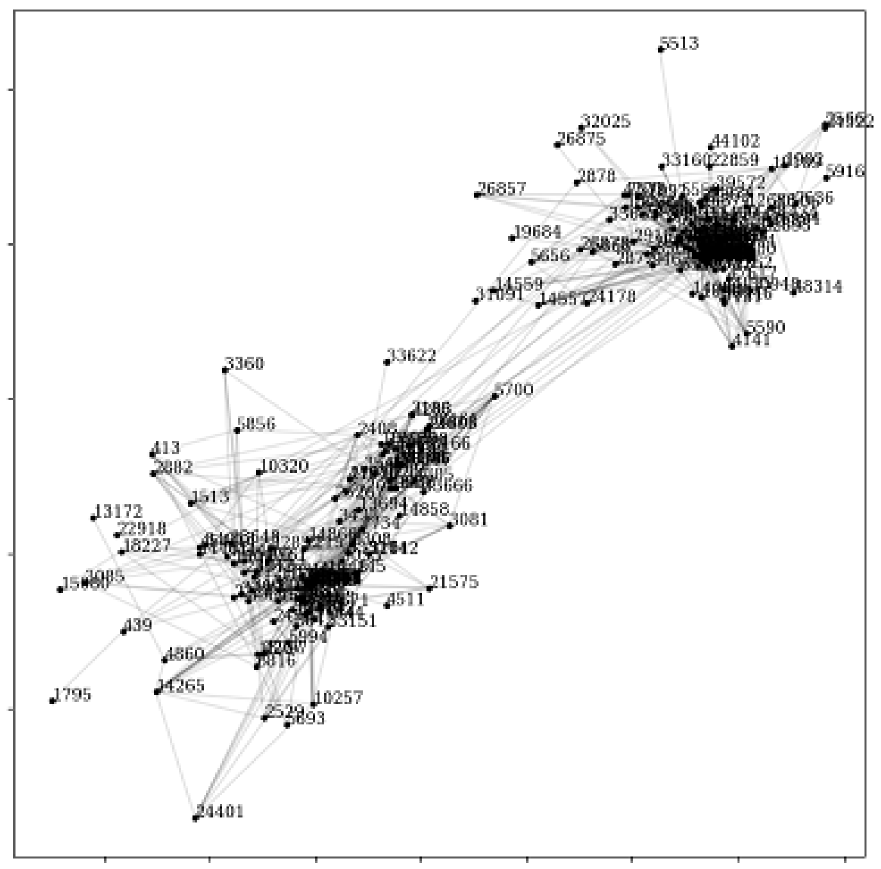


Figure 21: A 250-vertex subgraph of the Brightkite social network [49] containing vertices within a 50-mile radius of Washington, D.C. and Philadelphia, PA.

considering edge addition only within the local neighborhood of each pixel. Graph theoretic methods can then be applied to partition the resulting graph representation of the image.

In our experiment, we do not restrict edges to local neighborhoods. This more global approach to edge addition allows the possibility for a object to include distant pixels; that is, we do not assume that an object must be composed of a contiguous set of pixels (i.e., an object could be disjoint). To improve tractability, we construct a non-weighted graph in which edges are added between vertices if the “distance” between their associated pixel colors is within some threshold, which is a decreasing linear function of geometric distance. This approach requires two notions of distance. For geometric distance, we use the Euclidean

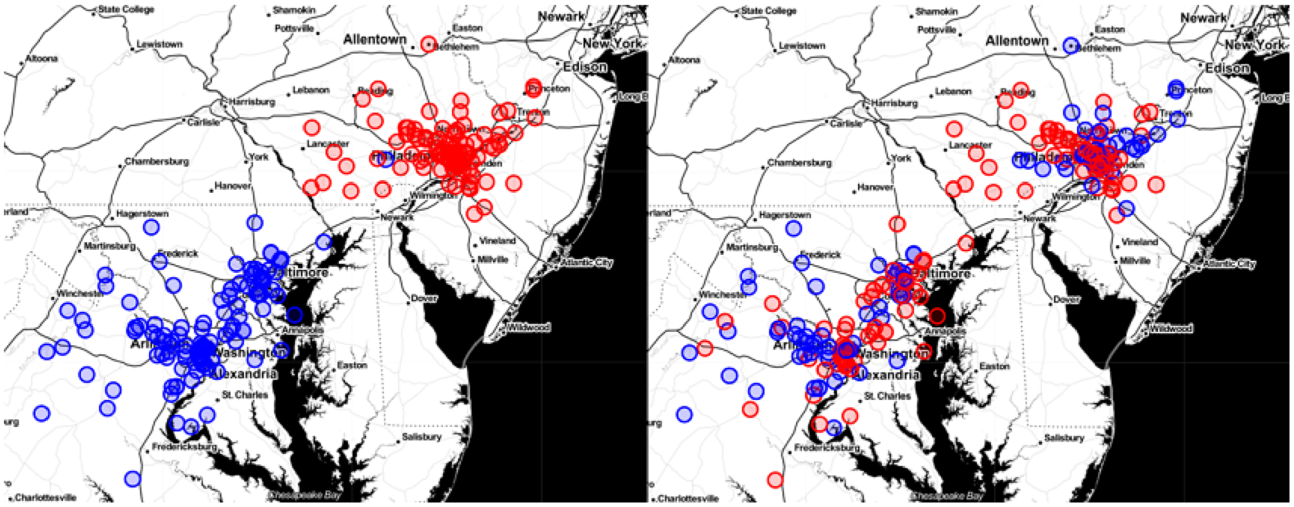


Figure 22: Communities identified by spectral partitioning (left) and Algorithm 4-6 (right).

distance between pixel location indices. For the distance between colors, we use the Python colormath library (specifically the color_diff.delta_e_cie2000 method) by [77]. To further improve tractability for larger images, our procedure considers edge addition at only some probability for pixel pairs whose geometric distance exceeds some threshold. Once the graph is constructed, we can apply Algorithm 4-6 to generate an image partition.

Figure 23 depicts the partition generated by Algorithm 4-6 for a heatmap image. We note that our method of graph construction allows the resulting segmented objects to be disjoint. A disjoint partition would be desirable if, for example, the heat map were topographic and the goal were to distinguish between high elevation and low elevation regions.

We now consider an image of cervical cells depicted in the left panel of Figure 24. In the female body, the cervix is a channel-like organ that connects the uterus and vagina. A Pap smear is a cervical cancer screening test, which involves removing a small quantity of cells from the surface of the cervix. The specimen is then plated onto a microscope slide and stained to visually enhance certain cell features, such as the nucleus and cytoplasm, important for early detection of precancerous cells. Human review of each microscope slide, which can contain thousands of cells, is tedious and labor-intensive. As such, there have been many efforts to automate review of Pap smear slides (see for example the survey of [8]).

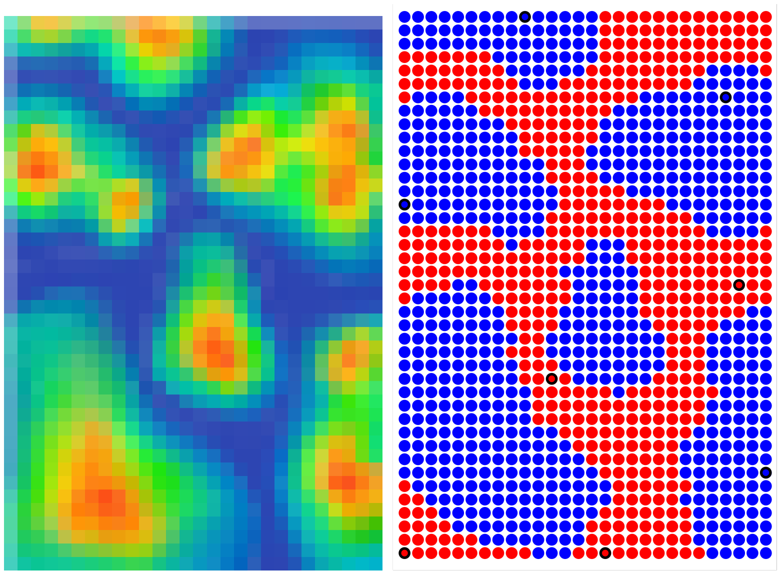


Figure 23: Partitioning of a heatmap by Algorithm 4-6.

A recent example of an automated Pap smear slide review process is proposed in [85]. The process consists of image acquisition and pre-processing, segmentation of individual cells from the whole slide image, feature extraction and selection to characterize each cell, and finally a machine learning classification model to identify potentially precancerous cells. The most predictive features in the classification model included the ratio of the cell's nucleus area to cytoplasm area, the green and red color intensities of the nucleus, and the green and red color intensities of the cytoplasm. Note that cytoplasm color is an indicator of the cell's originating location on the cervix ("superficial" or "intermediate").

In our experiment, we filter out the slide background and cell nucleai using a threshold technique, such that the resulting image contains only cell cytoplasm. The right panel of Figure 24 depicts that Algorithm 4-6 generates a similar partition in which the red cells are assigned to one object and the green cells are assigned to the second object. As previously noted, our method of constructing the graph allows the resulting segmented objects to be disjoint, which is desirable in this example of partitioning images of cells. The partitioning technique could fit into the automated process of [85], for example, as a pre-processing step to separate the cells by their originating location ("superficial" or "intermediate"). Then, two classification models could be trained (one for red cells and the other for green), rather than introducing that complexity into one classification model as was done by [85].

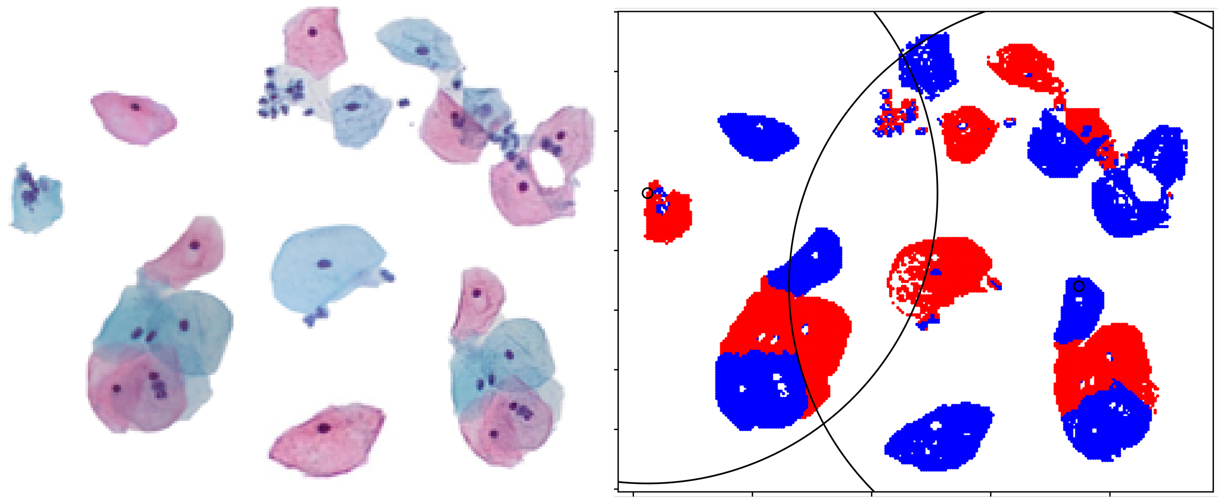


Figure 24: Partitioning of cells in a pap smear cytology slide by Algorithm 4-6.

4.10 Discussion

We considered the problem of recovering the ground-truth communities of a vertex-attributed graph that are *locally distinguishable* in the sense that the ground-truth communities exhibit stronger community structure in more local parts of the graph (i.e., in subgraphs induced by vertices whose attributes are closer). As we demonstrated in Subsection 4.2.1, the problem arises when clustering training data that contains responses generated from different latent sources. More specifically, we showed that the ground-truth communities (corresponding to the latent sources) of certain *similarity graphs* for the training data are locally distinguishable under mild conditions. Locally distinguishable communities, however, are not necessarily *globally distinguishable* (i.e. have higher internal than external connection density), posing a challenge for standard community detection methods. Accordingly, in Sections 4.2.2 and 4.4-4.7, we proposed and studied a community detection algorithm that first clusters the vertex set into *local* vertex subsets, then applies a spectral community detection algorithm to each graph induced by one of the local vertex subsets to obtain *local* communities, and finally agglomerates the local communities into *global* communities. In Theorem 4.3.1 we established conditions under which the algorithm *almost exactly* recovers communities when applied to the *local stochastic block model* (LSBM), a generalization of the stochastic block model (SBM) that we propose in Section 4.3 that incorporates the local distinguishability property. In Section 4.8 we evaluated the empirical performance of the method on LSBM instances and its viability for clustering training data. We observed that our method outperforms a spectral method benchmark when ground-truth communities are locally but not globally distinguishable, as we would expect. Furthermore, the method's performance can significantly improve when a small subset of labeled data is available.

There are a number of interesting directions for future work. First, naturally it is of interest to develop exact (instead of almost exact) recovery guarantees. Second, in the context of clustering training data, it might make more sense to consider an alternative to the local spectral community detection method (i.e., Algorithm 4-3), particularly a method designed for low-dimensional clustering, given that clustering points with similar features more or less boils down to clustering their one-dimensional responses. Third, and we think most importantly, there is a need for a better agglomeration method; one direction is to develop an optimization-based agglomeration method. Furthermore, if partially labeled data is available, naturally it is of interest to exploit this when agglomerating. Finally, it is of interest to explore using our method for community detection problems that arise in applications other than clustering training data.

4.11 Analysis of Algorithm 4-2

Here we set out to prove Theorem 4.4.1 and 4.4.2. First we introduce the notion of an ϵ-net (with respect to the Euclidean metric d_2) for a set $X \subseteq \mathbb{R}^m$.

Definition 4.11.1. For $\epsilon > 0$ and $X \subseteq \mathbb{R}^m$, we say that a set $\mathcal{N} \subseteq X$ is an *ϵ-net for X* (with respect to the Euclidean metric d_2) if for each $y \in X$ there exists $z \in \mathcal{N}$ such that $d_2(y, z) \leq \epsilon$.

If $X \subseteq [0,1]^m$, then there is an ϵ-net for X of cardinality $|\mathcal{N}| \leq \left(\frac{2\sqrt{m}}{\epsilon}\right)^m$:

Lemma 4.11.1. *Let $X \subseteq [0,1]^m$. For $0 < \epsilon < 1$, there is an ϵ-net $\mathcal{N}$ for X of cardinality $|\mathcal{N}| \leq \left(\frac{2\sqrt{m}}{\epsilon}\right)^m$.*

Proof. For $\zeta \in \mathbb{Z}_{>0}$, it is straightforward to see that there is a subset $\mathcal{N}' \subset [0,1]^m$ of cardinality $|\mathcal{N}'| = \zeta^m$ such that the collection $\left\{B_\infty\left(y, \frac{1}{2\zeta}\right)\right\}_{y \in \mathcal{N}'}$ partitions $[0,1]^m$. Because $B_\infty\left(y, \frac{1}{2\zeta}\right) \subseteq B_2\left(y, \frac{\sqrt{m}}{2\zeta}\right)$ for each $y \in \mathcal{N}'$, it follows that the collection $\left\{B_2\left(y, \frac{\sqrt{m}}{2\zeta}\right)\right\}_{y \in \mathcal{N}'}$ covers $[0,1]^m$. So, taking $\zeta = \left\lceil \frac{\sqrt{m}}{\epsilon} \right\rceil$, we have that there is a subset $\mathcal{N}_1 \subset [0,1]^m$ of cardinality $|\mathcal{N}_1| = \left\lceil \frac{\sqrt{m}}{\epsilon} \right\rceil^m \leq \left(\frac{2\sqrt{m}}{\epsilon}\right)^m$ such that the collection $\left\{B_2\left(y, \frac{\sqrt{m}}{2\zeta}\right)\right\}_{y \in \mathcal{N}_1}$ covers $[0,1]^m$. Because $B_2\left(y, \frac{\sqrt{m}}{2\zeta}\right) \subseteq B_2\left(y, \frac{\epsilon}{2}\right)$ for each $y \in \mathcal{N}_1$, the collection $\left\{B_2\left(y, \frac{\epsilon}{2}\right)\right\}_{y \in \mathcal{N}_1}$ also covers $[0,1]^m$. Let $\mathcal{N}_2 = \{y \in \mathcal{N}_1 : B_2\left(y, \frac{\epsilon}{2}\right) \cap X \neq \emptyset\}$. For each $y \in \mathcal{N}_2$, take $z_y \in B_2\left(y, \frac{\epsilon}{2}\right) \cap X$. Then the set $\mathcal{N} = \{z_y\}_{y \in \mathcal{N}_2}$ is an ϵ-net for X. $\square$

With Lemma 4.11.1 in hand, we are now prepared to prove Theorem 4.4.1:

Proof of Theorem 4.4.1. From Proposition 4.11.1 there is a $\frac{r}{2}$-net $\mathcal{N}$ for $X = \{x_i\}_{i \in [n]}$ of cardinality $|\mathcal{N}| \leq \left(\frac{4\sqrt{m}}{r}\right)^m \leq k$. It follows that

$$\min_{y_1, \ldots, y_k \in X} \max_{i \in [n]} \min_{j \in [k]} d_2(x_i, y_j) \leq \frac{r}{2}. \tag{4-9}$$

In other words, the optimal value of the k-center problem is less than or equal to $\frac{r}{2}$. Because the greedy algorithm of [34] is a 2-approximation for the k-center problem, the set U of center indices that Algorithm 4-2 constructs satisfies

$$\max_{i \in [n]} \min_{j \in U} d_2(x_i, x_j) \leq 2 \min_{y_1, \ldots, y_k \in X} \max_{i \in [n]} \min_{j \in [k]} d_2(x_i, y_j) \leq r,$$

where the second inequality follows from (4-9). Thus, the local vertex subsets cover V. □

Before proving Theorem 4.4.2, consider Lemma 4.11.2. The lemma states that every ball in a collection of balls centered at points in an ϵ-net for $[0,1]^m$ contains at least $\frac{\gamma(\epsilon)}{2}n$ attributes with high probability. The proposition follows from a Chernoff bound and the union bound.

Lemma 4.11.2. *Suppose that $x_i \sim \mathcal{U}$, $i \in [n]$ are independent. Let $0 < \epsilon < 1$ and $\mathcal{N}$ be any ϵ-net for $[0,1]^m$. Then*

$$\mathbb{P}\left(|B(y,\epsilon) \cap \{x_i\}_{i\in[n]}| \geq \frac{\gamma(\epsilon)}{2}n,\ \forall y \in \mathcal{N}\right) \geq 1 - |\mathcal{N}| \exp\left(-\frac{\gamma_{\mathcal{D}}(\epsilon)\, n}{8}\right).$$

Proof. Let $y \in \mathcal{N}$. The random variable $N_y := |B(y,\epsilon) \cap \{x_i\}_{i\in[n]}|$ follows a binomial distribution with parameter n and success probability parameter $p_y = \mathbb{P}(x_i \in B(y,\epsilon)) \geq \gamma(\epsilon)$, where the inequality follows from the definition of $\gamma(\epsilon)$. Hence

$$\mathbb{P}\left(N_y \leq \frac{\gamma(\epsilon)}{2}n\right) \leq \mathbb{P}\left(N_y \leq \frac{p_y}{2}n\right) \leq \exp\left(-\frac{p_y n}{8}\right) \leq \exp\left(-\frac{\gamma(\epsilon) n}{8}\right), \tag{4-10}$$

where the second inequality follows from a standard Chernoff bound for the binomial distribution. Now observe that

$$\begin{aligned}\mathbb{P}\left(N_y \geq \frac{\gamma(\epsilon)}{2}n,\ \forall y \in \mathcal{N}\right) &= 1 - \mathbb{P}\left(\exists y \in \mathcal{N} : N_y \leq \frac{\gamma(\epsilon)}{2}n\right)\\ &\geq 1 - \sum_{y\in\mathcal{N}} \mathbb{P}\left(N_y \leq \frac{\gamma(\epsilon)}{2}n\right)\\ &\geq 1 - |\mathcal{N}| \exp\left(-\frac{\gamma(\epsilon)\, n}{8}\right),\end{aligned}$$

where the first inequality follows from the union bound, and the second inequality follows from (4-10). □

Now we prove Theorem 4.4.2:

Proof of Theorem 4.4.2. By Lemma 4.11.1, there is a $\frac{r}{8}$-net $\mathcal{N}$ of cardinality $|\mathcal{N}| \leq \left(\frac{16\sqrt{m}}{r}\right)^m$ for $[0,1]^m$. Suppose that $|B(y, \frac{r}{8}) \cap \{x_i\}_{i\in[n]}| \geq \frac{1}{2}\gamma\left(\frac{\lambda}{8}\right) n$ for all $y \in \mathcal{N}$, which happens with probability at least $1 - \left(\frac{16\sqrt{m}}{r}\right)^m \exp\left(-\frac{\gamma(r/8)n}{8}\right)$ by Proposition 4.11.2 together with $|\mathcal{N}| \leq \left(\frac{16\sqrt{m}}{r}\right)^m$. It is sufficient to show that the $\frac{\gamma(r/8)}{2}$-intersection graph of the local vertex subsets $V_1, \ldots, V_k$ is connected.

Let U denote the set of center indices that Algorithm 4-2 constructs. From Theorem 4.4.1, it follows that the collection $\{B\left(x_i, \frac{r}{4}\right)\}_{i\in U}$ covers $\{x_i\}_{i\in[n]}$. (While we run Algorithm 4-2 with input r, not $r/4$, the algorithm only uses r after the set U is constructed.) So, because $B(y, \frac{r}{8}) \cap \{x_i\}_{i\in[n]} \neq \emptyset$ for all $y \in \mathcal{N}$, the collection $\{B\left(x_i, \frac{r}{2}\right)\}_{i\in U}$ covers $\cup_{y\in\mathcal{N}} B_2\left(y, \frac{r}{8}\right) \supseteq [0,1]^m$. Consider the graph H that has vertex set U and an edge between vertices $i, j \in U$ if $B\left(x_i, \frac{r}{2}\right) \cap B\left(x_j, \frac{r}{2}\right) \neq \emptyset$. We see that H is connected because the collection $\{B\left(x_i, \frac{r}{2}\right)\}_{i\in U}$ covers $[0,1]^m$ and $\{x_i\}_{i\in U} \subseteq [0,1]^m$. Let $i, j \in U$ such that there is an edge between i and j in H. To show that the $\frac{1}{2}\gamma\left(\frac{r}{8}\right)$-intersection graph of the local vertex subsets $V_1, \ldots, V_k$ is connected, it is sufficient to show that $B\left(x_i, r\right) \cap B\left(x_j, r\right)$ contains at least $\frac{1}{2}\gamma\left(\frac{r}{8}\right)$ of the attributes. By definition of H, there exists $y \in B\left(x_i, \frac{r}{2}\right) \cap B\left(x_j, \frac{r}{2}\right)$. Take $z \in \mathcal{N}$ such that $z \in B\left(y, \frac{r}{8}\right)$. Then

$$B\left(y, \frac{r}{8}\right) \subset B\left(z, \frac{r}{2}\right) \subseteq B\left(x_i, r\right) \cap B\left(x_j, r\right),$$

and hence $B\left(x_i, r\right) \cap B\left(x_j, r\right)$ contains at least $\frac{1}{2}\gamma\left(\frac{r}{8}\right)$ attributes as $B\left(y, \frac{r}{8}\right)$ contains at least this amount of attributes. □

4.12 Analysis of Algorithm 4-3

Our main goal in this section is to establish Theorem 4.5.1. Let us recall the setup of Section 4.5. Let $x \in \mathbb{R}^{n\times m}$ and $C^* = (C_1^*, C_2^*)$ be global ground-truth community pair. Suppose that $V' \subseteq V$ is a $\frac{1}{4}$-local vertex subset with respect to x of cardinality $n' = |V'|$. Without loss of generality, we assume that $V' = [n']$. Further suppose that $A' \in \mathbb{R}^{n'\times n'}$ is generated according to (4-5) under x and C^*. Let $\tilde{P} := \mathbb{E}[\tilde{A}]$, where $\tilde{A}$ is defined as in (4-4). In the proofs that we present in this section, we will use the definitions of P', u, and D from Section 4.5 along with the expression for $\mathbb{E}[\tilde{A}] = \tilde{P}$ in (4-6).

Our analysis follows in the footsteps of existing analysis for related spectral methods applied to the SBM. The main challenge is that this existing analysis heavily relies on analytic formulas for the eigenvectors and eigenvalues of $\tilde{P}$ that do not generalize to the context of the LSBM. Accordingly, we first establish properties of the eigenvalues and eigenvectors of $\tilde{P}$ in Lemma 4.12.1 and 4.12.2 below, respectively.

We introduce the following notation. Let $\lambda_1(M) \geq \lambda_2(M) \geq \cdots \geq \lambda_{n'}(M)$ denote the eigenvalues of a symmetric matrix $M \in \mathbb{R}^{n' \times n'}$. For each $i \in [n']$, define $u_i(M)$ to be an eigenvector of M with eigenvalue $\lambda_i(M)$. Finally let $\|M\|_F$ and $\|M\|_2$ denote the Frobenius and operator norm of M, respectively.

Lemma 4.12.1 presents a lower bound on $\lambda_1(\tilde{P})$ and an upper bound on $\lambda_2(\tilde{P})$; these bounds will ultimately enable us to lower bound the difference $\lambda_1(\tilde{P}) - \lambda_2(\tilde{P})$.

Lemma 4.12.1. *It holds that*

1. $\lambda_1(\tilde{P}) \geq \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) n'$ *and*

2. $\lambda_2(\tilde{P}) \leq (\alpha+\beta)\lambda n'$.

Proof. First we define the sets

$$\mathcal{S}_1 := \{(i,j) \in V' \times V' : i \neq j \text{ and either } i,j \in C_1^* \text{ or } i,j \in C_2^*\},$$
$$\mathcal{S}_2 := \{(i,j) \in V' \times V' : i \neq j \text{ and either } i \in C_1^*,\ j \in C_2^* \text{ or } i \in C_2^*,\ j \in C_1^*\}.$$

Note that $\mathcal{S}_1$, $\mathcal{S}_2$, and $\{(i,i)\}_{i \in V'}$ partition $V' \times V' = [n']^2$.

Let us establish the first statement of the lemma. Observe that

$$\begin{aligned}
&\lambda_1(\tilde{P}) \\
&= \max_{\|x\|_2=1} x^\top \tilde{P} x \\
&\geq \left(\frac{1}{\sqrt{n'}}u\right)^\top \tilde{P} \left(\frac{1}{\sqrt{n'}}u\right) \\
&= \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) n' + \frac{1}{n} u^\top D u \\
&= \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) n' + \frac{1}{n'} \sum_{i,j\in[n']} D_{ij} u_i u_j \\
&= \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) n' + \frac{\alpha}{n'} \sum_{(i,j)\in\mathcal{S}_1} \left(\lambda - \frac{d_2(x_i,x_j)}{\sqrt{m}}\right) + \frac{\beta}{n'} \sum_{(i,j)\in\mathcal{S}_2} \left(\lambda - \frac{d_2(x_i,x_j)}{\sqrt{m}}\right) \\
&\geq \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) n',
\end{aligned}$$

where the first inequality follows from $\|u\|_2 = \sqrt{n}$, the second equality follows from (4-6), the fourth equality follows from the definitions of u and D, and the last inequality follows from the fact that V' is a λ-local vertex subset.

Now let us show the second statement of the lemma. Because $|D_{ij}| \leq \lambda\alpha$ for $(i,j) \in \mathcal{S}_1$, and $|D_{ij}| \leq \lambda\beta$ for $(i,j) \in \mathcal{S}_2$, we have that

$$\|D\|_F \leq \lambda \left(\sum_{(i,j)\in\mathcal{S}_1} \alpha^2 + \sum_{(i,j)\in\mathcal{S}_1} \beta^2 \right)^{1/2} \leq \lambda n' \sqrt{\alpha^2+\beta^2} \leq (\alpha+\beta)\lambda n'. \tag{4-11}$$

From (4-6) and Weyl's inequality,

$$\lambda_2(\tilde{P}) \leq \left(\frac{p-q+(\alpha+\beta)(1-\lambda)}{2}\right) \lambda_2\left(uu^\top\right) + \lambda_1(D) \leq \|D\|_F \leq (\alpha-\beta)\lambda n',$$

where the second inequality follows from $\lambda_2(uu^\top) = 0$ and $\lambda_1(D) \leq \|D\|_F$, and the last inequality from (4-11). $\square$

Lemma 4.12.2 shows that the signs of the entries of $u_1(\tilde{P})$ capture the ground-truth communities. The lemma also establishes a lower bound on the magnitude of the entries.

Lemma 4.12.2. *Suppose that $p-q>0$ or $\alpha>\beta$ and $\lambda<1$. Then the following statements hold.*

1. *Each entry of* $u_1(\tilde{P})$ *is nonzero.*
2. *For* $i, j \in [n']$, *the sign of* $(u_1(\tilde{P}))_i$ *is different from the sign of* $(u_1(\tilde{P}))_j$ *if* $i \in C_1^*$, $j \in C_2^*$ *or* $i \in C_2^*$, $j \in C_1^*$.
3. $\min_{i\in[n']}|(u_1(\tilde{P}))_i| \geq \frac{1-\lambda}{2\sqrt{n'}}$.

Proof. Let $\tilde{M} = \operatorname{diag}(u)\tilde{P}\operatorname{diag}(u)$. The matrix $\tilde{M}$ has the same eigenvalues as $\tilde{P}$, and $u_1(\tilde{M}) = \operatorname{diag}(u)u_1(\tilde{P})$ is an eigenvector of $\tilde{M}$ with eigenvalue $\lambda_1(\tilde{P})$ (i.e., the largest eigenvalue of $\tilde{M}$). As discussed in Section 4.5, all entries of $\tilde{M}$ are positive under the assumption that $p - q > 0$ or $\alpha > \beta$ and $\lambda < 1$. Consequently, from the Perron-Frobenius theorem, $\operatorname{diag}(u)u_1(\tilde{P})$ has either all positive or all negative entries. It follows that the first and second statement of the lemma hold.

To establish the third statement, it is sufficient to show that $\min_{i\in[n']}(u_1(\tilde{M}))_i \geq \frac{1-\lambda}{2\sqrt{n'}}$. The main result of [56] implies that

$$\min_{i\in[n']}(u_1(\tilde{M}))_i \geq \left(\frac{\min_{i,j\in[n']}\tilde{M}_{ij}}{\max_{i,j\in[n']}\tilde{M}_{ij}}\right)\max_{i\in[n']}(u_1(\tilde{M}))_i. \tag{4-12}$$

We bound the right hand side of (4-12) from below. Because $\|u_1(\tilde{M})\|_2 = 1$,

$$\max_{i\in[n']}(u_1(\tilde{M}))_i \geq \frac{1}{\sqrt{n'}}. \tag{4-13}$$

From (4-6) and the definition of $\tilde{M}$,

$$\begin{aligned}\min_{i,j\in[n']}\tilde{M}_{ij} &= \frac{p-q+(\alpha+\beta)(1-\lambda)}{2} + \min_{i,j\in[n']}|D_{ij}| \\ &\geq \frac{p-q+(\alpha+\beta)(1-\lambda)}{2} \\ &\geq \frac{(1-\lambda)(p-q+\alpha+\beta)}{2}.\end{aligned} \tag{4-14}$$

Similarly, we have that

$$\begin{aligned}\max_{i,j\in V'} \tilde{M}_{ij} &= \frac{p-q+(\alpha+\beta)(1-\lambda)}{2} + \max_{i,j\in[n']}|D_{ij}| \\ &\le \frac{p-q+(\alpha+\beta)(1-\lambda)}{2} + \max\{\alpha,\beta\}\lambda \\ &= \frac{p-q+\alpha+\beta}{2} + \frac{(2\max\{\alpha,\beta\}-(\alpha+\beta))\lambda}{2} \\ &\le \frac{p-q+\alpha+\beta}{2} + \frac{(\alpha+\beta)\lambda}{2} \\ &\le p-q+\alpha+\beta. \qquad (4\text{-}15)\end{aligned}$$

The second statement follows from (4-12) together with (4-13), (4-14), and (4-15). □

We are now prepared to prove Theorem 4.5.1. The proof utilizes the version of the Davis-Kahan theorem from [93] (specifically, Corollary 3). In our setting, the theorem tell us that surely

$$\min_{s\in\{-1,1\}}\|u_1(\tilde{A}) - su_1(\tilde{P})\|_2 \le \frac{2^{3/2}\|\tilde{A}-\tilde{P}\|_2}{\lambda_1(\tilde{P})-\lambda_2(\tilde{P})}. \qquad (4\text{-}16)$$

We also use the following concentration inequality for the operator norm of $A' - P'$:

$$\mathbb{P}(\|A'-P'\|_2 \le \hat{c}_1\sqrt{n'}) \ge 1 - \exp(-\hat{c}_2 n'), \qquad (4\text{-}17)$$

where $\hat{c}_1, \hat{c}_2 > 0$ are absolute constants. The inequality holds because the entries of A' are (up to symmetry) independent Bernoulli random variables and $P' = \mathbb{E}[A']$; see, for example, Section 2.3 of [76].

Proof of Theorem 4.5.1. From Lemma 4.12.2 and $\lambda \le 1/4$, we have that $\min_{i\in[n']}|(u_1(\tilde{P}))_i| \ge \frac{3}{8\sqrt{n'}}$. Hence, surely

$$\begin{aligned}1 - A(C, C^*) &\le \frac{1}{n}\min_{s\in\{-1,1\}}\left\|\frac{8}{3}\sqrt{n}(u_1(\tilde{A}) - su_1(\tilde{P}))\right\|_2^2 \\ &= \left(\frac{8}{3}\min_{s\in\{-1,1\}}\|u_1(\tilde{A}) - su_1(\tilde{P})\|_2\right)^2 \\ &\le \left(\frac{8}{3}\cdot\frac{2^{3/2}\|\tilde{A}-\tilde{P}\|_2}{\lambda_1(\tilde{P})-\lambda_2(\tilde{P})}\right)^2 \\ &= \left(\frac{8}{3}\cdot\frac{2^{3/2}\|A'-P'\|_2}{\lambda_1(\tilde{P})-\lambda_2(\tilde{P})}\right)^2\end{aligned}$$

where the second inequality follows from (4-16). It follows from (4-17) that there are absolute constants $\tilde{c}_1, c_2 > 0$ such that

$$\begin{aligned}
&1 - \exp(-c_2 n') \\
&\leq \mathbb{P}\left(1 - A(C, C^*) \leq \left(\frac{\tilde{c}_1\sqrt{n'}}{\lambda_1(\tilde{P}) - \lambda_2(\tilde{P})}\right)^2\right) \\
&\leq \mathbb{P}\left(1 - A(C, C^*) \leq \left(\frac{\tilde{c}_1\sqrt{n'}}{((p-q)/2 + (\alpha+\beta)3/8 - (\alpha+\beta)/4)n'}\right)^2\right) \\
&= \mathbb{P}\left(1 - A(C, C^*) \leq \left(\frac{\tilde{c}_1}{((p-q)/2 + (\alpha+\beta)/8)\sqrt{n'}}\right)^2\right) \\
&\leq \mathbb{P}\left(1 - A(C, C^*) \leq \left(\frac{8\tilde{c}_1}{(p-q+\alpha+\beta)\sqrt{n'}}\right)^2\right) \\
&= \mathbb{P}\left(1 - A(C, C^*) \leq \frac{c_1}{D_\ell^2 n'}\right),
\end{aligned}$$

where the second inequality follows from Lemma 4.12.1 and $\lambda \leq 1/4$, and the last equality follows from the definition of D_ℓ and taking $c_1 = 64\tilde{c}_1^2$. □

4.13 Analysis of Algorithm 4-4

Here we prove Theorem 4.6.1. Let us recall the setup of Section 4.6. Let $G = (V, E, x)$ be a vertex-attributed graph that has ground-truth community pair $C^* = (C_1^*, C_2^*)$. Also let $C^{(i)} = (C_1^{(i)}, C_2^{(i)})$, $i \in [k]$ be local community pairs. Assume that each vertex in V belongs to at least one of the local communities. Recall that $V^{(i)} = C_1^{(i)} \cup C_2^{(i)}$ for $i \in [k]$, and define $n_i := |V^{(i)}|$. Also recall the definition of $C^{(i \setminus j)}$ for $i \neq j \in [k]$.

Consider any two local community pairs, say $C^{(1)}$ and $C^{(2)}$. Lemma 4.13.1 shows that if we "appropriately" agglomerate $C^{(1)}$ and $C^{(2)}$, then we will not lose too much agreement. That is, the second condition of the lemma posits (without loss of generality) that for each $\ell \in [2]$, it holds that $C_1^{(\ell)}$ and $C_2^{(\ell)}$ share more vertices with the ground-truth communities C_1^* and C_2^*, respectively, than with C_2^* and C_1^*, respectively. So it is appropriate to agglomerate $C_1^{(1)}$

with $C_1^{(2)}$ and agglomerate $C_2^{(1)}$ with $C_2^{(2)}$, i.e. to construct new agglomerated communities $C_1^{(3)} = C_1^{(1)} \cup C_1^{(2\setminus 1)}$ and $C_2^{(3)} = C_2^{(1)} \cup C_2^{(2\setminus 1)}$.

Lemma 4.13.1. *Let $0 \leq \delta \leq \frac{1}{2}$. For each $\ell \in [2]$, suppose that*

1. *$A(C^{(\ell)}, C^*) \geq 1 - \delta$ and*
2. *$A(C^{(\ell)}, C^*) = \frac{1}{n_\ell}(|C_1^{(\ell)} \cap C_1^*| + |C_2^{(\ell)} \cap C_2^*|)$.*

Let $C_1^{(3)} = C_1^{(1)} \cup C_1^{(2\setminus 1)}$ and $C_2^{(3)} = C_2^{(1)} \cup C_2^{(2\setminus 1)}$. Then for $C^{(3)} = (C_1^{(3)}, C_2^{(3)})$, we have

$$A(C^{(3)}, C^*) \geq 1 - 2\delta.$$

Proof. Let $n_{1\wedge 2} = |V^{(1)} \cap V^{(2)}|$. Also let $V^{(3)} = V^{(1)} \cup V^{(2)}$ and $n_3 = |V^{(3)}|$. Note that $n_3 = n_1 + n_2 - n_{1\wedge 2}$.

From the definition of $A(C^{(3)}, C^*)$,

$$\begin{aligned}
A(C^{(3)}, C^*) &\geq \frac{1}{n_3}(|C_1^{(3)} \cap C_1^*| + |C_2^{(3)} \cap C_2^*|) \\
&= \frac{1}{n_3}(|(C_1^{(1)} \cup C_1^{(2\setminus 1)}) \cap C_1^*| + |(C_2^{(1)} \cup C_2^{(2\setminus 1)}) \cap C_2^*|) \\
&= \frac{1}{n_3}(|C_1^{(1)} \cap C_1^*| + |C_1^{(2\setminus 1)} \cap C_1^*| + |C_2^{(1)} \cap C_2^*| + |C_2^{(2\setminus 1)} \cap C_2^*|) \\
&= \frac{1}{n_3}(|C_1^{(1)} \cap C_1^*| + |C_1^{(2)} \cap C_1^*| - |C_1^{(2)} \cap V^{(1)} \cap C_1^*|) \\
&+ \frac{1}{n_3}(|C_2^{(1)} \cap C_2^*| + |C_2^{(2)} \cap C_2^*| - |C_2^{(2)} \cap V^{(1)} \cap C_2^*|) \\
&= \frac{1}{n_3}(n_1 A(C^{(1)}, C) + n_2 A(C^{(2)}, C) - |C_1^{(2)} \cap V^{(1)} \cap C_1^*| - |C_2^{(2)} \cap V^{(1)} \cap C_2^*|) \\
&\geq \frac{1}{n_3}((1-\delta)(n_1 + n_2) - |C_1^{(2)} \cap V^{(1)}| - |C_2^{(2)} \cap V^{(1)}|) \\
&= \frac{1}{n_3}((1-\delta)(n_3 + n_{1\wedge 2}) - |V^{(1)} \cap V^{(2)}|) \\
&= \frac{1}{n_3}((1-\delta)n_3 - \delta n_{1\wedge 2}) \\
&\geq 1 - 2\delta
\end{aligned}$$

where the fourth equality follows from the second condition of the lemma, the second inequality follows from the first condition of the lemma, and the last inequality follows from $n_{1\wedge 2} \leq n_3$. □

Next we show in Lemma 4.13.2 that if the agreement of local community pairs $C^{(1)}$ and $C^{(2)}$ is sufficiently high (relative to the number of vertices that they intersect on), then vertex agglomeration appropriately agglomerates them. Condition (4-18) ensures that Algorithm 4-4 selects the appropriate local communities to agglomerate together in Step 4.

Lemma 4.13.2. *Suppose that the following conditions hold.*

1. $|V^{(1)} \cap V^{(2)}| \geq \eta$.
2. *For* $\ell \in [2]$, *it holds that* $A(C^{(\ell)}, C^*) \geq 1 - \frac{\eta}{5n}$ *and*
3. $A(C^{(\ell)}, C^*) = \frac{1}{n_\ell}(|C_1^{(\ell)} \cap C_1^*| + |C_2^{(\ell)} \cap C_2^*|)$.

Then

$$|C_1^{(1)} \cap C_1^{(2)}| + |C_2^{(1)} \cap C_2^{(2)}| > |C_1^{(1)} \cap C_2^{(2)}| + |C_2^{(1)} \cap C_1^{(2)}|. \tag{4-18}$$

Proof. For the sake of contradiction, suppose that

$$|C_1^{(1)} \cap C_1^{(2)}| + |C_2^{(1)} \cap C_2^{(2)}| \leq |C_1^{(1)} \cap C_2^{(2)}| + |C_2^{(1)} \cap C_1^{(2)}|. \tag{4-19}$$

Now observe that

$$\begin{aligned}
\eta &\leq |(C_1^{(1)} \cup C_2^{(1)}) \cap (C_1^{(2)} \cup C_2^{(2)})| \\
&= |C_1^{(1)} \cap C_1^{(2)}| + |C_1^{(1)} \cap C_2^{(2)}| + |C_2^{(1)} \cap C_1^{(2)}| + |C_2^{(1)} \cap C_2^{(2)}| \\
&\leq 2(|C_1^{(1)} \cap C_2^{(2)}| + |C_2^{(1)} \cap C_1^{(2)}|) \\
&= 2(|C_1^{(1)} \cap C_2^{(2)} \cap C_1^*| + |C_1^{(1)} \cap C_2^{(2)} \cap C_2^*| + |C_2^{(1)} \cap C_1^{(2)} \cap C_1^*| + |C_2^{(1)} \cap C_1^{(2)} \cap C_2^*|) \\
&\leq 2(|C_2^{(2)} \cap C_1^*| + |C_1^{(1)} \cap C_2^*| + |C_2^{(1)} \cap C_1^*| + |C_1^{(2)} \cap C_2^*|) \\
&= 2(n_1(1 - A(C^{(1)}, C^*)) + n_2(1 - A(C^{(2)}, C^*))) \\
&\leq 2(\frac{n_1\eta}{5n} + \frac{n_2\eta}{5n}) \\
&\leq \frac{4\eta}{5}
\end{aligned}$$

where the second inequality follows from (4-19), the fourth inequality from the second condition of the lemma, and the last inequality from $n_1, n_2 \leq n$. We have obtained a contradiction. □

We are now prepared to prove Theorem 4.6.1:

Proof of Theorem 4.6.1. It is sufficient to show that after the ℓ-th agglomeration for $0 \leq \ell \leq k-1$, it holds that

$$\min_{i \in U} A(C^{(i)}, C^*) \geq 1 - \frac{\zeta\eta}{2^{k+1-\ell}n}. \tag{4-20}$$

We proceed by induction. Clearly (4-20) holds for $\ell = 0$, so suppose it holds for $0 \leq \ell < k-1$. Then at the beginning of the iteration $\ell + 1$ of Algorithm 4-4, we have that

$$\min_{i \in U} A(C^{(i)}, C^*) \geq 1 - \frac{\zeta\eta}{2^{k+1-\ell}n} \geq 1 - \frac{\zeta\eta}{8n} \geq 1 - \frac{\zeta\eta}{5n},$$

where the second inequality follows from $\ell < k-1$. Lemma 4.13.2 implies that the appropriate communities are agglomerated together in iteration $\ell + 1$. It follows from Lemma 4.13.1 that after the $(\ell+1)$-th agglomeration,

$$\min_{i \in U} A(C^{(i)}, C^*) \geq 1 - \frac{2\zeta\eta}{2^{k+1-\ell}n} = 1 - \frac{\zeta\eta}{2^{k+1-(\ell+1)}n}.$$

Thus, by induction, the desired result holds. □

4.14 Analysis of Algorithm 4-5

Proof of Theorem 4.7.1. We break the proof into four steps. The first three correspond to the three main steps of Algorithm 4-5, and the last combines the results from the first three together.

Step 1: Local vertex subset construction. Let $\eta = \frac{\gamma(1/32)n}{2}$. We introduce the event

$$E_1 = \left\{ \cup_{i \in [k]} V^{(i)} = V \text{ and } \eta\text{-intersection graph of } V^{(i)}, i \in [k] \text{ is connected} \right\}.$$

Because $k = \lceil (64\sqrt{m})^m \rceil \geq (16\sqrt{m})^m = \left(\frac{4\sqrt{m}}{r}\right)^m$ (as $r = \frac{1}{4}$), we have from Theorem 4.4.1 that $\cup_{i \in [k]} V^{(i)} = V$ surely. Accordingly,

$$\begin{aligned} \mathbb{P}(E_1) &= \mathbb{P}(\eta\text{-intersection graph of } V^{(i)}, i \in [k] \text{ is connected}) \\ &\geq 1 - \left(64\sqrt{m}\right)^m \exp\left(-\frac{\gamma(1/32)n}{8}\right), \\ &\geq 1 - \hat{c}_1(m)\exp(-\hat{c}_2(m)n), \end{aligned} \tag{4-21}$$

where the first inequality follows from Theorem 4.4.2 with $r = \frac{1}{4}$, and the second equality follows from taking $\hat{c}_1(m) = (64\sqrt{m})^m$ and $\hat{c}_2(m) \le \frac{\gamma(1/32)}{8}$. (Such a constant $\hat{c}_2(m)$ indeed exists; see Section 4.4.)

Step 2: Local community detection. For each $i \in [k]$, consider the event

$$E_{2i} = \left\{ A(C^{(i)}, C^*) \ge 1 - \frac{\hat{c}_3}{D_\ell^2 \eta} \right\},$$

where $\hat{c}_3$ equals the absolute constant c_1 in Theorem 4.5.1. Because $V^{(i)}$ is a $\frac{1}{4\sqrt{m}}$-local vertex subset and $E_1 \subseteq \{|V^{(i)}| \ge \eta\}$, we can apply Theorem 4.5.1 to obtain that

$$\mathbb{P}(E_{2i} \mid E_1) \ge 1 - \exp(-\hat{c}_4 \eta), \tag{4-22}$$

where $\hat{c}_4$ equals the absolute constant c_2 in the theorem. For

$$E_2 = \left\{ \min_{i\in[k]} A(C^{(i)}, C^*) \ge 1 - \frac{\hat{c}_3}{D_\ell^2 \eta} \right\},$$

we can apply (4-22) and the union bound to obtain that

$$\begin{aligned} \mathbb{P}(E_2 \mid E_1) &\ge 1 - k \exp(-\hat{c}_4 \eta) \\ &\ge 1 - (65\sqrt{m})^m \exp\left(-\frac{\hat{c}_4 \gamma(1/32) n}{2} \right) \\ &= 1 - \hat{c}_5(m) \exp(-\hat{c}_6(m) n), \end{aligned} \tag{4-23}$$

where the second inequality follows from $k \le (65\sqrt{m})^m$ and the definition of η, and the equality from taking $\hat{c}_5(m) = (65\sqrt{m})^m$ and $\hat{c}_6(m) \le \frac{\hat{c}_4 \gamma(1/32)}{2}$.

Step 3: Agglomeration. Let us take

$$c_1(m) = \frac{\hat{c}_3^{1/2} 2^{(\lceil (64\sqrt{m})^m \rceil + 3)/2}}{\gamma(1/32)} = \frac{\hat{c}_3^{1/2} 2^{(k+3)/2}}{\gamma(1/32)},$$

and define

$$\zeta = \frac{c_1(m)^2}{D_\ell^2 n}.$$

First note that from (4-7) that $\zeta \le 1$. Second note that from the definitions of ζ and η,

$$E_2 = \left\{ \min_{i\in[k]} A(C^{(i)}, C^*) \ge 1 - \frac{\zeta \eta}{2^{k+1} n} \right\}.$$

Accordingly, for $E_3 = \left\{A(C, C^*) \geq 1 - \frac{\zeta\eta}{4n}\right\}$, we can apply Theorem 4.6.1 to obtain that

$$\mathbb{P}(E_3 \mid E_1 \cap E_2) = 1. \tag{4-24}$$

Step 4: Putting it all together. Finally, take

$$c_2(m) = \frac{\hat{c}_3 2^{(64\sqrt{m})^m}}{\gamma(1/32)} = \frac{\hat{c}_3 2^k}{\gamma(1/32)}.$$

Then we have that $E_3 = \left\{A(C, C^*) \geq 1 - \frac{c_2(m)}{D_\ell^2 n}\right\}$. Observe that

$$\begin{aligned}
\mathbb{P}(E_3) &\geq \mathbb{P}(E_1 \cap E_2 \cap E_3) \\
&= \mathbb{P}(E_3 \mid E_1 \cap E_2)\mathbb{P}(E_2 \mid E_1)\mathbb{P}(E_1) \\
&\geq 1 - \hat{c}_1(m)\exp(-\hat{c}_2(m)n) - \hat{c}_5(m)\exp(-\hat{c}_6(m)n),
\end{aligned}$$

where the second inequality follows from (4-21), (4-23), and (4-24). Thus the desired result follows. □

5.0 Conclusions

We have studied and developed algorithms for spatial variations of three classic optimization problems. In Table 2, we summarize each algorithmic contribution with respect to our intention that they be implementation-friendly, scalable, and performance-guaranteed.

Table 2: Summary of our algorithmic contributions with respect to the intended traits

Algorithmic Contribution	**Implementation-friendly**	**Scalable**	**Performance-guaranteed**
Minimizing makespan across preloaded machines	Does not require optimization expertise	$n \log n$ complexity enables scaling	2-approximation for identical machines
The meet-in-the-middle k-center problem	Requires solving the standard k-center problem along with a simple assignment algorithm	Polynomial complexity enables scaling	Guarantees provided that are specific to each cost regime
Recovering locally distinguishable communities with applications to clustering training data	Requires standard spectral partitioning along with a decomposition and agglomeration scheme	Decomposition and agglomeration enables scaling	Guarantee provided for instances of the LSBM

Finally, we offer a few interesting directions for future research in the following paragraphs.

Minimizing makespan across preloaded machines. While we consider minimizing makespan, in practice jobs may be prioritized by some measure of importance. A modified version of Algorithm 2-1 that minimizes makespan while satisfying job priority constraints would be a meaningful extension. Analysis of the modified algorithm might also be performed to establish a performance guarantee. In light of the fact that the list scheduling algorithm of Graham [35] admits a $\frac{4}{3} - \frac{1}{3m}$ approximation guarantee, it is natural to wonder if the modified version of Algorithm 2-1 considered in Theorem 2.3.2 (i.e., that uses an ordered list in Step 'a' of Stage 2) also admits a factor $\frac{4}{3} - \frac{1}{3m}$ approximation guarantee. It would also be interesting to develop a more general model along with solution algorithms for problems with machine-specific ready and delivery times. Finally, it would be interesting to consider uncertain processing times, as well as settings in which transportation is constrained.

The meet-in-the-middle $k-$center problem. One could consider the meet-in-the-middle capability in the context of other facility location problems, such as those with fixed costs, supplies, demands, and capacities. Our meet-in-the-middle problem could be extended to consider facilities that deliver goods to a sequence (route) of locations from which nearby clients retrieve goods by traveling. One could consider different objectives, such as minimizing the total travel distance, or minimizing the demand-weighted travel distance. Our problem, like all facility location problems, requires a choice of k as a user input. While a choice of k in practice may be determined by budget, methods that help the user determine an appropriate value (for example, by considering the cost of each additional facility in context with its incremental improvement to the objective value) would be meaningful. Finally, while we established that our analysis of Algorithm 3-2 is tight, one could explore whether other algorithms might provide better guarantees. Similarly, it would be interesting to explore algorithm performance under relaxed cost regimes, such as problems in which only Assumption 3.2.1 (i.e., metric costs) holds.

Recovering locally distinguishable communities with applications to clustering training data. First, developing exact (instead of almost exact) recovery guarantees would be a natural extension to our work. Second, in the context of clustering training data, it might be useful to consider methods designed for low-dimensional clustering, given that clustering points with similar features more or less reduces to clustering their one-dimensional responses. Third, and we think most importantly, there is a need for a better agglomeration method; one direction is to develop an optimization-based agglomeration method. Furthermore, if partially labeled data are available, naturally it is of interest to exploit this when agglomerating. Fourth, our community detection algorithm requires a choice of k as user input, which may be difficult in practice. Developing methods to help the user choose an appropriate value of k, especially when a partially labeled data are unavailable for semi-supervised learning, could improve algorithm performance in practice. Fifth, in healthcare and many other applications, each type of classification error (false positive and false negative) may have different consequences. Somehow considering the consequences of classification errors in the community detection algorithm might be a meaningful extension. Finally, one could explore using our method for community detection problems that arise in applications other than clustering training data.

Bibliography

[1] Emmanuel Abbe. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

[2] Emmanuel Abbe, Afonso S Bandeira, and Georgina Hall. Exact recovery in the stochastic block model. *IEEE Transactions on information theory*, 62(1):471–487, 2015.

[3] Emmanuel Abbe, Jianqing Fan, Kaizheng Wang, and Yiqiao Zhong. Entrywise eigenvector analysis of random matrices with low expected rank. *Annals of statistics*, 48(3):1452, 2020.

[4] Amir Ahmadi-Javid, Pardis Seyedi, and Siddhartha S. Syam. A survey of healthcare facility location. *Computers & Operations Research*, 79:223–263, 2017.

[5] Maria Albareda-Sambola, Yolanda Hinojosa, Alfredo Marín, and Justo Puerto. When centers can fail: A close second opportunity. *Computers & Operations Research*, 62:145–156, 2015.

[6] Aaron Archer. Two O(log*k)-approximation algorithms for the asymmetric k-center problem. In Karen Aardal and Bert Gerards, editors, *Integer Programming and Combinatorial Optimization*, pages 1–14, Heidelberg, 2001. Springer Berlin.

[7] Richard Becker, Allan Wilks, Ray Brownrigg, Thomas Minka, and Alex Deckmyn. *maps: Draw Geographical Maps*, 2022. R package version 3.4.1.

[8] Ewert Bengtsson and Patrik Malm. Screening for cervical cancer using automated analysis of pap-smears. *Computational and mathematical methods in medicine*, 2014, 2014.

[9] Peter J Bickel and Aiyou Chen. A nonparametric view of network models and newman–girvan and other modularities. *Proceedings of the National Academy of Sciences*, 106(50):21068–21073, 2009.

[10] Ernesto G Birgin and Débora P Ronconi. Heuristic methods for the single machine scheduling problem with different ready times and a common due date. *Engineering Optimization*, 44(10):1197–1208, 2012.

[11] Justine I Blanford, Supriya Kumar, Wei Luo, and Alan M MacEachren. It's a long, long walk: accessibility to hospitals, maternity and integrated health centers in Niger. *International Journal of Health Geographics*, 11(1):1–15, 2012.

[12] J. Blazewicz, K. Ecker, E. Pesch, G. G. Schmidt, M. Sterna, and J. Weglarz. *Handbook on Scheduling: From Theory to Practice*. Springer International Publishing AG, 2019.

[13] Ravi B Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 280–285. IEEE, 1987.

[14] Derya Celik Turkoglu and Mujde Erol Genevois. A comparative survey of service facility location problems. *Annals of Operations Research*, 292(1):399–468, 2020.

[15] O Chapelle, B Schölkopf, and A Zien. Semi-supervised learning. 2006.

[16] Yudong Chen and Jiaming Xu. Statistical-computational tradeoffs in planted problems and submatrix localization with a growing number of clusters and submatrices. *The Journal of Machine Learning Research*, 17(1):882–938, 2016.

[17] Zhi-Long Chen. Integrated production and outbound distribution scheduling: Review and extensions. *Operations Research*, 58(1):130–148, 2010.

[18] TC Edwin Cheng, Z-L Chen, and Natalia V Shakhlevich. Common due date assignment and scheduling with ready times. *Computers & Operations Research*, 29(14):1957–1967, 2002.

[19] Gene Cheung, Enrico Magli, Yuichi Tanaka, and Michael K Ng. Graph spectral image processing. *Proceedings of the IEEE*, 106(5):907–930, 2018.

[20] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090, 2011.

[21] David S Choi, Patrick J Wolfe, and Edoardo M Airoldi. Stochastic blockmodels with a growing number of classes. *Biometrika*, 99(2):273–284, 2012.

[22] Anne Condon and Richard M Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures & Algorithms*, 18(2):116–140, 2001.

[23] M.S. Daskin and L.K. Dean. Location of health care facilities. In *Operations Research and Health Care*, pages 43–76. Springer, 2004.

[24] Wayne S DeSarbo and William L Cron. A maximum likelihood methodology for clusterwise linear regression. *Journal of classification*, 5:249–282, 1988.

[25] Yash Deshpande, Subhabrata Sen, Andrea Montanari, and Elchanan Mossel. Contextual stochastic block models. *Advances in Neural Information Processing Systems*, 31, 2018.

[26] Martin E. Dyer and Alan M. Frieze. The solution of some random np-hard problems in polynomial expected time. *Journal of Algorithms*, 10(4):451–489, 1989.

[27] Luis Fanjul-Peyro and Rubén Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69, 2010.

[28] Reza Zanjirani Farahani, Samira Fallah, Rubén Ruiz, Sara Hosseini, and Nasrin Asgari. OR models in urban service facility location: A critical review of applications and future developments. *European Journal of Operational Research*, 276(1):1–27, 2019.

[29] Jessica M Franklin, Chandrasekar Gopalakrishnan, Alexis A Krumme, Karandeep Singh, James R Rogers, Joe Kimura, Caroline McKay, Newell E McElwee, and Niteesh K Choudhry. The relative benefits of claims and electronic health record data for predicting medication adherence trajectory. *American heart journal*, 197:153–162, 2018.

[30] Sainyam Galhotra, Arya Mazumdar, Soumyabrata Pal, and Barna Saha. The geometric block model. 32(1), 2018.

[31] Jesus Garcia-Diaz, Rolando Menchaca-Mendez, Ricardo Menchaca-Mendez, Saúl Pomares Hernández, Julio César Pérez-Sansalvador, and Noureddine Lakouari. Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation. *IEEE Access*, 7:109228–109245, 2019.

[32] Jesus Garcia-Diaz, Jairo Sanchez-Hernandez, Ricardo Menchaca-Mendez, and Rolando Menchaca-Mendez. When a worse approximation factor gives better performance: A 3-approximation algorithm for the vertex k-center problem. *Journal of Heuristics*, 23(5):349–366, 2017.

[33] Marco Ghirardi and Chris N Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005.

[34] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[35] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1969.

[36] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In P.L. Hammer, E.L. Johnson, and B.H. Korte, editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[37] W. Gu, X. Wang, and S.E. McGregor. Optimization of preventive health care facility locations. *International Journal of Health Geographics*, 9(1):17, 2010.

[38] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[39] Maryam Hasanzadeh Mofrad. *Optimizing vaccine clinic operations in low and middle income countries.* PhD thesis, University of Pittsburgh, 2016.

[40] Yuto Hashimoto and Henrique Ribeiro Carretti. Development and application of an immunization network design optimization model for UNICEF. 2020.

[41] Dorit S Hochbaum. When are NP-hard location problems easy? *Annals of Operations Research*, 1(3):201–214, 1984.

[42] Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[43] Georgia Spiridon Karanasiou, Evanthia Eleftherios Tripoliti, Theofilos Grigorios Papadopoulos, Fanis Georgios Kalatzis, Yorgos Goletsis, Katerina Kyriakos Naka, Aris

Bechlioulis, Abdelhamid Errachid, and Dimitrios Ioannis Fotiadis. Predicting adherence of patients with HF through machine learning techniques. *Healthcare technology letters*, 3(3):165–170, 2016.

[44] Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

[45] Jochen Könemann, Yanjun Li, Ojas Parekh, and Amitabh Sinha. An approximation algorithm for the edge-dilation k-center problem. *Operations Research Letters*, 32(5):491–495, 2004.

[46] Chung-Yee Lee and Zhi-Long Chen. Machine scheduling with transportation considerations. *Journal of scheduling*, 4(1):3–24, 2001.

[47] Chung-Yee Lee and Vitaly A Strusevich. Two-machine shop scheduling with an uncapacitated interstage transporter. *IIE transactions*, 37(8):725–736, 2005.

[48] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[49] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[50] J. Lim, E. Claypool, B.A. Norman, and Rajgopal J. Coverage models to determine outreach vaccination center locations in low and middle income countries. *Operations Research for Health Care*, 2016.

[51] Yang-Kuei Lin and Feng-Yu Hsieh. Unrelated parallel machine scheduling with setup times and ready times. *International Journal of Production Research*, 52(4):1200–1214, 2014.

[52] Ana Dolores López-Sánchez, Jesús Sánchez-Oro, and Alfredo García Hernández-Díaz. GRASP and VNS for solving the p-next center problem. *Computers & Operations Research*, 104:295–303, 2019.

[53] François Margot. *Symmetry in Integer Linear Programming*, pages 647–686. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[54] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, 1959.

[55] Frank McSherry. Spectral partitioning of random graphs. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 529–537. IEEE, 2001.

[56] Henryk Minc. On the maximal eigenvector of a positive matrix. *SIAM Journal on Numerical Analysis*, 7(3):424–427, 1970.

[57] Alan T Murray. A coverage model for improving public transit system accessibility and expanding access. *Annals of Operations Research*, 123(1):143–156, 2003.

[58] Sayed H Naderi, Jonathan P Bestwick, and David S Wald. Adherence to drugs that prevent cardiovascular disease: meta-analysis on 376,162 patients. *The American journal of medicine*, 125(9):882–887, 2012.

[59] Mark EJ Newman and Aaron Clauset. Structure and inference in annotated networks. *Nature communications*, 7(1):1–11, 2016.

[60] Abdisalan M Noor, Abdinasir A Amin, Peter W Gething, Peter M Atkinson, Simon I Hay, and Robert W Snow. Modelling distances travelled to government health services in Kenya. *Tropical Medicine & International Health*, 11(2):188–196, 2006.

[61] Camilo Ortiz-Astorquiza, Ivan Contreras, and Gilbert Laporte. Multi-level facility location problems. *European Journal of Operational Research*, 267(3):791–805, 2018.

[62] Rina Panigrahy and Sundar Vishwanathan. An O(log*n) approximation algorithm for the asymmetric p-center problem. *Journal of Algorithms*, 27(2):259–268, 1998.

[63] Bo Peng, Lei Zhang, and David Zhang. A survey of graph theoretical approaches to image segmentation. *Pattern recognition*, 46(3):1020–1038, 2013.

[64] C N Potts and V A Strusevich. Fifty years of scheduling: A survey of milestones. *Journal of the Operational Research Society*, 60(sup1):S41–S68, 2009.

[65] Chris N Potts. Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441, 1980.

[66] R. Quinlan. Auto MPG. UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5859H.

[67] S. Rahman and D.K. Smith. Use of location-allocation models in health service development planning in developing nations. *European Journal of Operational Research*, 123(3):437–452, 2000.

[68] A. Rais and A. Viana. Operations research in healthcare: A survey. *International Transactions in Operational Research*, 18(1):1–31, 2011.

[69] Ernesto Ramos and David Donoho. Statlib datasets archive. `http://lib.stat.cmu.edu/datasets/`, 1983. Accessed: 2023-09-21.

[70] Charles S ReVelle and Horst A Eiselt. Location analysis: A synthesis and survey. *European Journal of Operational Research*, 165(1):1–19, 2005.

[71] Charles S Revelle, Horst A Eiselt, and Mark S Daskin. A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research*, 184(3):817–848, 2008.

[72] Anthony A Saka. Model for determining optimum bus-stop spacing in urban areas. *Journal of Transportation Engineering*, 127(3):195–199, 2001.

[73] David B. Shmoys. Computing near-optimal solutions to combinatorial optimization problems. In *Combinatorial Optimization, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 355–397. AMS Publications, 1995.

[74] Tom AB Snijders and Krzysztof Nowicki. Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of classification*, 14(1):75–100, 1997.

[75] F. Tanser. Methodology for optimising location of new primary health care facilities in rural communities: A case study in KwaZulu-Natal, South Africa. *Journal of Epidemiology and Community health*, 60(10):846–850, 2006.

[76] Terence Tao. *Topics in random matrix theory*, volume 132. American Mathematical Society, 2023.

[77] Greg Taylor. python-colormath documentation. 2017.

[78] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

[79] J. Vandelaer, J. Bilous, and D. Nshimirimana. Reaching every district (RED) approach: A way to improve immunization performance. *Bulletin of the World Health Organization*, 86(3):A–B, 2008.

[80] Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

[81] V. Verter and S.D. Lapierre. Location of preventive health care facilities. *Annals of Operations Research*, 110(1-4):123–132, 2002.

[82] Sundar Vishwanathan. An O(log* n) approximation algorithm for the asymmetric p-center problem. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–5. SIAM, 1996.

[83] Van Vu. A simple svd algorithm for finding hidden partitions. *Combinatorics, Probability and Computing*, 27(1):124–140, 2018.

[84] Ran Wei, Xiaoyue Liu, Yongjian Mu, Liming Wang, Aaron Golub, and Steven Farber. Evaluating public transit services for operational efficiency and access equity. *Journal of Transport Geography*, 65:70–79, 2017.

[85] Kyi Pyar Win, Yuttana Kitjaidure, Kazuhiko Hamamoto, and Thet Myo Aung. Computer-assisted screening for cervical cancer using digital image processing of pap smear images. *Applied Sciences*, 10(5):1800, 2020.

[86] Gerhard J Woeginger. Heuristics for parallel machine scheduling with delivery times. *Acta Informatica*, 31(6):503–512, 1994.

[87] World Health Organization and others. In-depth evaluation of the reaching every district approach in the African region. `https://cdn.who.int/media/docs/default-source/immunization/reaching-every-district-red/1-afro-1-red-evaluation-report-2007-final.pdf`, 2007. Online; accessed June 13, 2023.

[88] Zhiqiang Xu, Yiping Ke, Yi Wang, Hong Cheng, and James Cheng. A model-based approach to attributed graph clustering. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, pages 505–516, 2012.

[89] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th international conference on data mining*, pages 1151–1156. IEEE, 2013.

[90] Tianbao Yang, Rong Jin, Yun Chi, and Shenghuo Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 927–936, 2009.

[91] Yuwen Yang and Jayant Rajgopal. Learning combined set covering and traveling salesman problem. *preprint, arXiv:2007.03203*, 2020.

[92] Yuwen Yang and Jayant Rajgopal. Outreach strategies for vaccine distribution: A multi-period stochastic modeling approach. *Operations Research Forum*, 2(2), apr 2021.

[93] Yi Yu, Tengyao Wang, and Richard J Samworth. A useful variant of the davis–kahan theorem for statisticians. *Biometrika*, 102(2):315–323, 2015.

[94] Jinjiang Yuan, Ameur Soukhal, Youjun Chen, and Lingfa Lu. A note on the complexity of flow shop scheduling with transportation constraints. *European Journal of Operational Research*, 178(3):918–925, 2007.

[95] Marjan Zakeri, Sujit S Sansgiry, and Susan M Abughosh. Application of machine learning in predicting medication adherence of patients with cardiovascular diseases: a systematic review of the literature. *Journal of Medical Artificial Intelligence*, 5(5):1–16, 2022.

[96] Hugo Zanghi, Stevenn Volant, and Christophe Ambroise. Clustering based on random graph model embedding vertex features. *Pattern Recognition Letters*, 31(9):830–836, 2010.