

Optimal network locality in distributed virtualized data-centers

Jimmy Leblet, Zhe Li, Gwendal Simon and Di Yuan

Linköping University Post Print

N.B.: When citing this work, cite the original article.

Original Publication:

Jimmy Leblet, Zhe Li, Gwendal Simon and Di Yuan, Optimal network locality in distributed virtualized data-centers, 2011, Computer Communications, (34), 16, 1968-1979.

<http://dx.doi.org/10.1016/j.comcom.2011.06.002>

Copyright: Elsevier

<http://www.elsevier.com/>

Postprint available at: Linköping University Electronic Press

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-72030>

Optimal Network Locality in Distributed Virtualized Data-Centers

Jimmy Leblet^{c,b}, Zhe Li^{c,2,*}, Gwendal Simon^c, Di Yuan^{a,1}

^a*Linköping University - Linköping, Sweden*

^b*Université Lyon 3 - Lyon, France*

^c*Institut Telecom, Telecom Bretagne - Brest, France*

Abstract

Cost efficiency is a key aspect in deploying distributed service in networks within decentralized service delivery architectures. In this paper, we address this aspect from an optimization and algorithmic standpoint. The research deals with the placement of service components to network sites, where the performance metric is the cost for acquiring components between the sites. The resulting optimization problem, which we refer to as the k -Component Multi-site Placement Problem, is applicable to service distribution in a wide range of communication networking scenarios. We provide a theoretical analysis of the problem's computational complexity, and develop an integer programming model for providing reference results for performance benchmarking. On the algorithmic side, we present four approaches: an algorithm with approximation guarantee and three heuristics algorithms. The first heuristic is derived from graph theory on domatic partition. The second heuristic, built on intuition, admits distributed computation. The third heuristic emphasizes on fairness in cost distribution among the sites. We report simulation results for sets of networks where cost is represented by round-trip time (RTT) originating from real measurements. For small networks, the integer model is used to study algorithm performance in terms of optimality. Large networks are used to compare the al-

*Corresponding author

¹The work of this author is supported by CENIIT, Linköping University, Sweden

²The work of this author is supported by Vipeer project and Agence Nationale de la Recherche, France

gorithms relatively to each other. Among the algorithms, the heuristic based on intuition has close-to-optimal performance, and the fairness heuristic achieves a good balance between single-site cost and the overall one. In addition, the experiments demonstrate the significance of optimization for cost reduction in comparison to a the random allocation strategy.

Keywords: distributed services, network locality, optimization problem, peer-to-peer, approximation algorithm

1. Introduction

In the age of cloud computing, a service provider ensures the massive and fast delivery of its service by replicating it on a large set of *sites* (servers, data-centers, *etc.*). For example, more than 30 data-centers are used by the Google search engine [1]. The platforms for deploying web-based services such as Amazon Elastic Computing Cloud (EC2) [2] and Microsoft Azure [3] use also this *horizontal scaling* approach based on a high number of cheap commodity servers to provide scalability to the service providers. In the case of popular services like Facebook or Twitter, whose size is in the order of hundreds of Terabytes, the service is not fully replicated on every site. Instead, the idea is to partition the service (data and links among data) into smaller *components*, and then to locate the components on distinct sites [4]. The principle of such *Distributed Virtualized Data-Centers* is detailed in [5]. In online social networking, the service is randomly partitioned using a Distributed Hash Table (DHT). The number of partitions and the number of sites are industrial choices of the service provider.

In the meantime, several concurrent studies have recently called for an even more decentralized approach of horizontal scaling [6, 7, 8, 9, 10]. The motivation comes from the observation that large-scale data-centers are difficult to manage [9, 11]. Hence, some works have claimed that significant gains can be obtained from an architecture consisting of a larger set of sites, whose size ranges from container-sized data-centers to set-top-boxes [6, 12]. In this context, even small-size services need to be partitioned and deployed over multiple sites.

In this paper, we address from an optimization standpoint the problem of locating a set of service components on a set of sites managed by a service provider. In general terms, we consider a service composed by k distinct components hosted by n sites, where n can be large. We focus on the management of *distant* sites in a context where the network distance between them is the prevalent issue. As our subsequent discussions will frequently rely on graph terminology, the terms, site, node, and vertex will be used interchangeably. For simplicity, we assume that each site can host only one component. We remark however that a site being able to physically host multiple components can be modeled as several distinct sites at the same location. All sites act as front-end servers to clients. That is, every site receives and treats the requests of some of the clients. It is possible that the component hosted by the site is enough to give a response, but, more probably, the other service components have to be reached. We consider that the service has been randomly partitioned, which is the strategy currently used by most service providers, *e.g.* Facebook [13], so each site will *uniformly* access the $k - 1$ components that they do not have. Our goal is to minimize the overall *network cost* for accessing the remote components. The network cost for any pair of sites i and j is a generic notion that may include the number of Autonomous Systems traversed by a packet from i to j , the round-trip delay time or the ecological impact implied by the load of all routers on the route. For convenience, the cost for a pair of sites is also referred to as the distance between them. The network cost for a given site i corresponds to the sum of the costs between i and the $k - 1$ sites that are closest in cost/distance and host collectively the $k - 1$ remote components. The goal is to minimize the total of these costs over the sites. We will refer to this problem as the *k -Component Multi-Site Placement Problem (k -CMSP)*.

The work presented in this paper consists in the following investigations. We first provide a thorough theoretical analysis of this optimization problem. In particular we prove that, in the general case, k -CMSP is NP-complete. As an immediate consequence, two related optimization problems, where every site may be allocated several components, and where only a subset of all components

are requested by each site are both NP-complete as well. Next, we formulate k -CMSP by means of an Integer Programming model. Solving the model gives exact solutions of k -CMSP, which is doable in small networks. These two investigations provide a scientific background for extensive research on k -CMSP.

Our research is largely driven by practical considerations. Thus the next part of our investigation has consisted in designing several algorithms that can be easily implemented by a service provider. We propose four algorithms, one approximation algorithm and three heuristics. The first algorithm is a $(\frac{3}{2}k - \frac{5}{2})$ -approximation algorithm that is inspired by a recent work on a variant of the Facility Location Problem [14]. Our algorithm ensures that resulting total cost is no more than $\frac{3}{2}k - \frac{5}{2}$ times the optimal one. Our second algorithm, theoretically elegant but complex, originates from domination theory in graphs. The algorithm considers the nearest neighbor graph linking the closest sites together, followed by partitioning this graph into k distinct dominating sets, each being the sites allocated the same component. In order to obtain fast a deterministic graph partition, we use a technique performing graph augmentation. The heuristic gives a solution in which each site is likely to find all components among its nearest neighbors. The third algorithm, being intuitive and far simpler, considers also the nearest neighbor graph, and each site examines its two-hop neighbors to decide its own component allocation. Finally, the last heuristic aims to build a fair solution where no site is outrageously far from all components. The idea is to rank the total cost for each site and consider the most disadvantaged site first.

We present a generic set of simulations. For small networks, we report performance evaluation of the algorithms using the global optimum computed via the integer programming model. For large networks, the algorithms are compared to each other, and vis-a-vis a random allocation strategy. The main insight is that the intuitive heuristic appears to have near-optimal performances for k -CMSP, whereas the ranking heuristic provides convincing results in terms of fairness. Moreover, the very basic random allocation that is suggested in current studies is largely outperformed. These results show that implementing

quick algorithms may provide a significant improvement in the service delivery performance metric of k -CMSP; the improvement can be translated into less latency for clients and less cost for network operators. We finally outline future works that will follow up the theoretical investigations and the experimental results of the paper.

2. Related Works

The problem addressed in this paper is closely related to two well-known families of theoretical problems: Facility Location Problems (FLP) and κ -median problems (κ MP). We detail both families in the following.

In the general (uncapacitated) FLP, we have a set of facilities and a set of clients. An opening cost is associated with every facility. Each client has an access cost to retrieve the service deployed at any facility. The objective is to open a subset of the facilities and to satisfy all demands of the clients so that the total cost is minimized. This problem has been widely considered in networks to locate replica of web content into caches. Various constant-factor approximation algorithms apply on a metric space. The algorithmic concepts can be based on LP-rounding [15, 16] or on primal-dual approach [17, 18]. The 1.52 approximation ratio that has been obtained in most recent studies is close to the lower bound proved in [19]. Many variants of FLP have been studied; most of them are NP-hard [20]. The variant being closest to our problem is the k -Product UFLP (k -PUFLP), where each client needs to be supplied with k distinct products. Few papers have dealt with this problem, though, and the focus in the available references is classic optimal and heuristic algorithms [21, 22, 23]. The heuristic algorithms proposed in [21, 22] are, in fact, techniques to improve the computation of the integer program. Most recently, a first approximate solution has been proposed, where the solution is at most $\frac{3}{2}k - 1$ times the optimal one [14]. To some extent, the k -CMSP can be seen as a special case of k -PUFLP as it is possible to transform any instance of k -CMSP into an instance of k -PUFLP. Indeed, every site can be seen as both a facility and a client with a null access

cost between them. The approximation algorithm we describe later is a variant of the algorithm presented in [14], but the approximation ratio in our case is better. Note that, because k -CMSP is a special case, the NP-completeness proof of k -PUFLP does not apply directly to k -CMSP. Therefore we provide a thorough investigation of the NP-completeness of k -CMSP in this paper.

In κ MP, we do not consider the opening cost of a facility but the number of opened facilities is restricted to κ . Various approximation algorithms using a variety of techniques have been proposed: rounding of linear programs [24], primal-dual methods [25] and local search [26, 27]. In [26] the authors proposed an approximation algorithm for κ MP using at most $(3 + 5/\epsilon) \cdot \kappa$ facilities and giving a total cost that is at most $1 + \epsilon$ times the optimal solution of at most κ facilities. This result is improved in [27] by a $3 + 2/p$ -approximation algorithm with a time complexity of $O(n^p)$, where p is the number of facilities that can be changed in one swap operation. The main difference between k -CMSP and κ MP is that we do not limit the number of sites (facilities) serving one component, but on the other hand each site (client) must access to k sites including itself to get all the service components. If we regard the allocation of each component as an independent κ -median problem, then deciding the value of κ for any component becomes another combinatorial problem since every site can hold only one component. Thus, the problem we consider cannot be just treated as several independent κ MPs. Note also that, in both κ MP and FLP, the demand of a node (on the single type of service) is given, whereas in k -CMSP, every node has to access all the service components except the one allocated to itself. Hence which components a site has to fetch remotely depends on the allocation decision at the site.

In [28], service allocation in the Internet is modeled as both κ MP and FLP. The authors proposed an efficient distributed algorithm, and demonstrated that the algorithm performs close to optimality even when information gathering is restricted to a small local environment. As a consequence of the differences in problem characteristics, their algorithm, based on solving small κ MPs or FLPs within local environments, does not directly apply to our problem. In particular,

the merge operation required by the algorithm, which performs aggregation and estimation of the demand of sites outside a local environment, does not admit any immediate generalization to k -CMSP.

Another problem related to k -CMSP is the replication placement problem in P2P networks. Many replication techniques are reported in the last decade. Uniform, proportional and square root replications are proposed in [29, 30, 31]. These techniques distribute the replication either uniformly throughout the network or by the query rate of the objects. In [32, 33, 34], the authors presented several schemes based on peer selection policy, *e.g.*, random selection of peers, and selecting peers on the path from the providing peer to the requesting peers. Some dynamic replication schemes are given in [35, 36]. These replication techniques aim at finding a good trade-off between redundancy and cost-efficiency of resources. The dynamics of peers is a main issue in P2P networks. The redundancy is used to deal with peer churn, but excessive replication causes wastage of peer resources. However, in our context, all the sites are under the control of a service provider, thus peer churn is not considered as a critical problem. Moreover, most of the replication schemes for P2P systems are only effective for locating popular objects. But in our system, each component in the application is uniformly accessed. Thus, the replication techniques in P2P systems are not appropriate to k -CMSP.

The resource placement in Content Delivery Network (CDN) is also a related problem to k -CMSP. The resource allocation problem in CDN can be divided into two categories: edge server deployment and content replication placement. The former can be solved using traditional solutions for FLP [37] or κ MP [38], and is out of the scope of the current paper. For content replication placement, several strategies have been investigated in [39]. Most of these strategies replicate the entire content object on edge servers, therefore the choice of the replicated content becomes critical for the performance of CDN, and it is usually associated with the popularity of the content. In our optimization framework, one content object is cut into several components. Since each component of an object is uniformly accessed, the placement of component is independent of the

popularity of the content. Thus, k -CMSP can be regarded as a new approach to the replication placement in CDN, or a supplementary study in this domain.

3. Problem Analysis

In this part, we provide an analysis of k -CMSP, including a proof showing that the corresponding decision problem is NP-complete. That is, unless $P = NP$, this optimization problem can not be reasonably solved for large instances (here for a large number of sites). Then, we develop an integer programming model. For small networks, the integer model can be used to find exact solutions of k -CMSP. This allows for gaining insights into the performance of other algorithms, particularly distributed algorithms, in terms of optimality.

As a preamble, we recall some definitions of domination theory that will be used throughout the paper. Consider a graph G linking the system elements, the node set is \mathcal{V} . A set $V \subseteq \mathcal{V}$ is called a *dominating set* if, for every node $i \in \mathcal{V}$, i is either an element of V or is adjacent to at least one element of V . Thus, if the nodes in V (called dominators) are all allocated the same component, then the component becomes available within one hop for all sites. An extension of dominating set is a *domatic partition*, where the graph G is partitioned into several distinct dominating sets. If the dominating distance of a dominator is extended to s hops, we call the set of dominators an *s -dominating set*. Given a graph G and a positive integer K , the problem of determining if there exists a domatic partition or s -domatic partition of size K is NP-complete [40, 41].

3.1. NP-Completeness

In the following, we show that the decision problem associated to k -CMSP is NP-complete. We recall that n denotes the number of sites and k is the number of components.

k -Component Multi-Site Placement Problem

INSTANCE : A positive integer $n \in \mathbb{N}^*$, a positive integer $k \in \{1, \dots, n\}$, a distance function $d : \{1, \dots, n\} \times \{1, \dots, n\} \rightarrow \mathbb{R}^+$, where $d(i, i) = 0$ for any $i \in \{1, \dots, n\}$, and a positive integer $R \in \mathbb{N}$.

QUESTION : Is there an allocation $\varphi : \{1, \dots, n\} \rightarrow \{1, \dots, k\}$ such that the sum of all distances to other components is less than or equal to R , that is :

$$\sum_{i=1}^n \sum_{c=1}^k \min \{d(i, j) : j \in \{1, \dots, n\}, \varphi(j) = c\} \leq R$$

Theorem 1. *k-CMSP is NP-complete.*

PROOF. Given an instance of k -CMSP and a labeling φ , verifying that this function is valid can be clearly done in polynomial time in the size of the problem, hence k -CMSP belongs to NP. We assume that $R' > 1$ all over the proof.

We reduce the Domatic Partition problem to k -CMSP. Given a graph $G = (V, E)$ and a positive integer $R' \in \mathbb{N}^*$ where R' is the number of sets of the domatic partition, let n be the number of vertices of G (and assume that $V = \{1, \dots, n\}$), let $k = R'$, $d(i, j) = 1$ if $\{i, j\} \in E$ and $d(i, j) = R' \cdot n + 1$ otherwise, $d(i, i) = 0$, and $R = R' \cdot n$. Clearly the instance of the k -CMSP can be constructed in polynomial time in the size of the instance of the domatic partition. By construction, we obtain a special case of k -CMSP. If this special case is equivalent to the domatic partition instance (in terms of a yes/no answer), which is the bulk of the proof, then the general case of k -CMSP can not be easier than the NP-complete domatic partition problem. In the proof, we show that the two instances are indeed equivalent in the sense that there is a domatic partition of size R' (or k), if and only if the k -CMSP instance admits a valid mapping φ , i.e., the existence of a solution with total cost no more than $R = R' \cdot n = kn$.

For the forward implication, assume that G admits a domatic partition of size $k = R'$, for example V_1, \dots, V_k . Let φ be the function such that for any $x \in V$, we have $x \in V_{\varphi(x)}$, we show that φ is also a valid function for the instance of k -CMSP. Indeed, for any $i \in V$, by definition of φ and the fact that we have $d(i, j) \in \{1, R' \cdot n + 1\}$ for $i \neq j$, we obtain that for any $c \in \{1, \dots, k\} \setminus \varphi(i)$, $\min\{d(i, l) : l \in \{1, \dots, n\}, \varphi(l) = c\} = 1$. As a result,

$$\sum_{i=1}^n \sum_{\substack{c=1 \\ c \neq \varphi(i)}}^k \min_{l \in \{1, \dots, n\}} \{d(i, l) : \varphi(l) = c\} = n \cdot (k - 1) \leq R.$$

Now, as $d(i, i) = 0$, φ is a valid allocation for our problem.

For the backward implication, assume that φ is a valid function for the constructed instance of k -CMSP, and assume, by contradiction, that there does not exist a domatic partition of size k in G . This implies that there exists $i' \in V$ and $c' \in \{1, \dots, k\} \setminus \{\varphi(i')\}$ such that $\varphi^{-1}(c') \cap N_G(i') = \emptyset$, where $N_G(i')$ denotes the neighborhood of i' in G . Thus, by definition of $d(i', j)$, we obtain that $\min \{d(i', l) : l \in \{1, \dots, n\}, \varphi(l) = c'\} = R' \cdot n + 1$, which is strictly greater than R and hence $\sum_{i=1}^n \sum_{c=1, c \neq \varphi(i)}^k \min_{l \in \{1, \dots, n\}} \{d(i, l) : \varphi(l) = c\} = n \cdot (k - 1) \leq R' \cdot n + 1$ which is strictly greater than R and hence contradicts the definition of a valid allocation. \square

As an immediate consequence of Theorem 1, the more general problem, where each server can store l components for any integer $l \geq 1$, is also NP-complete, because k -CMSP corresponds to $l = 1$. A second observation is that the case where each client has to retrieve k' components with $k' \leq k$ is NP-complete as well.

3.2. Integer Programming

In order to solve k -CMSP, we first explore the domain of Integer Programming. k -CMSP can be formulated in form of an integer linear model using two sets of binary variables.

$$x_{ic} = \begin{cases} 1 & \text{if component } c \text{ is allocated at site } i \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ij}^c = \begin{cases} 1 & \text{if } i \text{ obtains component } c \text{ from } j \\ 0 & \text{otherwise} \end{cases}$$

Problem k -CMSP can be modeled as follows.

$$\begin{aligned}
& \text{Minimize } \sum_{c \in \mathcal{C}} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} d(i, j) y_{ij}^c \\
& \text{Subject to } \sum_{c \in \mathcal{C}} x_{ic} = 1, & \forall i \in \mathcal{V} & \quad (1) \\
& \sum_{j \neq i} y_{ij}^c = 1 - x_{ic}, & \forall i \in \mathcal{V}, \forall c \in \mathcal{C} & \quad (2) \\
& y_{ij}^c \leq x_{jc}, & \forall i, j \in \mathcal{V}, \forall c \in \mathcal{C} & \quad (3) \\
& x_{ic} \in \{0, 1\}, & \forall i \in \mathcal{V}, \forall c \in \mathcal{C} & \\
& y_{ij}^c \in \{0, 1\}, & \forall i, j \in \mathcal{V}, \forall c \in \mathcal{C} &
\end{aligned}$$

In the model, the set of components is noted \mathcal{C} . Thus $k = |\mathcal{C}|$. Constraint (1) ensures that every site will host exactly one component. Constraint (2) ensures that if site i does not obtain component c from any other site $j \neq i$, then i itself must have c . Conversely, if i is not allocated c , then i has to get c from exactly one of the other sites. Hence (2) is an equality, and implies, together with (1), that each site has access to all the components in \mathcal{C} . Constraint (3) states that i can obtain a component from another site j , only if that component is located at j .

Let $n = |\mathcal{V}|$. In the integer model, the number of variables and constraints equal $nk + n^2k$ and $n + nk + n^2k$, respectively. Since each site may potentially host any component, the number of x -variables can not be reduced. Variables $y_{ij}^c, c \in \mathcal{C}$ can be excluded from model, if the distance $d(i, j)$ is so large so it can not be optimal to deliver component from j to i . This requires however some a priori knowledge on the overall cost. From the fairness perspective, it may be of relevance to introduce a bound B_i on the service delivery time at each site i . The bound B_i constrains the distance to any site from which a component is delivered. To the integer model, the effect of the bound is size reduction, as all y -variables with $d(i, j) > B_i$ can be eliminated from consideration. On the other hand, too stringent bounds will lead to infeasibility. Examining the model constraints, the size of constraint (1) can be reduced by a factor of n , because it can be replaced by $\sum_{i \in \mathcal{V}} y_{ij}^c \leq nx_{jc}, \forall j, c$. From a computational efficiency

standpoint, the reduction is not preferred because it significantly weakens the bound from the linear programming relaxation, causing the size of the tree that has to be explored by branch-and-bound to grow by magnitudes.

For some network structures, the optimum solution may introduce unbalanced workload at the sites. Although load balancing is not within the scope of the current paper, we remark that additional constraints can be deployed to address this aspect. For example, we can add the following constraint:

$$\sum_{i \in \mathcal{V}} y_{ij}^c \leq L, \forall j, c.$$

By construction, the constraint imposes an upper bound L that restricts the number of sites served by a given site, in order to prevent site congestion.

4. Approximation Algorithm

Given an instance of k -CMSP, we first introduce a method to find a fractional optimal solution for the linear program of the integer model. We connect every site with its $k - 1$ nearest sites to form a $(k - 1)$ -nearest graph G_k . We denote the set of the $k - 1$ nearest sites of node i as $N_{G_k}(i) = \{j_1, j_2, \dots, j_{k-1}\}$ and $N_{G_k}[i] = N_{G_k}(i) \cup \{i\}$. We form the fractional optimal solution S by setting, for all i in \mathcal{V} and all c in \mathcal{C} , the value of x_{ic} to $\frac{1}{k}$, and, for all j in $N_{G_k}[i]$, the value of y_{ij}^c to $\frac{1}{k}$ too.

Lemma 2. *Solution S is a fractional optimal solution to k -CMSP.*

PROOF. It is obvious that S satisfies all the constraints in the model in Section 3.2, except the integrality requirement of the variables. Moreover, every node retrieves all the components from its $k - 1$ nearest sites and itself. Based on this observation, it is easily realized that there is no (fractional) solution with lower total cost, and hence S is an optimal fractional solution to k -CMSP. \square

We denote by $\bar{d}_i = \sum_{c=1}^k \sum_{j \in N_{G_k}[i]} d_{ij} \frac{1}{k}$ the cost of node i in the above fractional solution. For the approximation algorithm with output allocation φ (which corresponds to an integer solution to the model in the previous section),

let $d_i = \sum_{c=1}^k \min \{d_{ij} : j \in \{1, \dots, n\}, \varphi(j) = c\}$, and $F(i)$ be the set of nodes serving i , i.e., i collects the $k - 1$ absent components from the nodes in $F(i)$ in solution φ .

The algorithm contains two phases. Two sets are created and processed in algorithm execution: V_o and V_s . For all $i \in V_o$, the algorithm has determined in phase one the components for all nodes in $N_{G_k}[i]$. Set V_s contains nodes that have not been allocated any component in phase one. Both sets are empty when the algorithm starts. In the first phase, the algorithm iterates over the nodes. In one iteration, it selects the untreated node i with $\bar{d}_i = \min\{\bar{d}_i | i \in V\}$. If some of the neighbors in $N_{G_k}(i)$ have been allocated components, the algorithm checks the condition that *no pair of nodes in $N_{G_k}[i]$ hold the same component*. If $N_{G_k}[i]$ satisfies the condition, an allocation is performed to the remaining nodes in $N_{G_k}[i]$, such that no two nodes store the same component. As a result, all components appear in $N_{G_k}[i]$. Node i is then put in V_o . If $N_{G_k}[i]$ does not satisfy the condition, i is placed in V_s . The first phase terminates when all nodes have been processed. In the second phase, the algorithm considers nodes that are in V_s and have no component allocated. Each of these nodes is allocated the component that is farthest away in the current allocation. The algorithm is described in Algorithm 1.

We will now prove that this algorithm provides a guaranteed approximation of the optimal allocation. The proof requires the following lemma.

Lemma 3. *Any $(k - 1)$ -nearest graph admits a 3-domatic partition of size k .*

PROOF. In a 3-dominating partition, the nodes are divided into a number of subsets (k in our case), and every node can reach at least one node in each subset within 3 hops. We need to show here that, after the execution of Algorithm 1, every node can access all k components in at most 3 hops. If this holds, we can obviously view each set of nodes holding the same component as a 3-dominating set, therefore we would like to prove that Algorithm 1 produces a k -sized 3-domatic partition of G_k .

After the execution of the first phase of Algorithm 1 on G_k , a node is either

Algorithm 1: Approximation algorithm for k -CMSP

```
1  $V \leftarrow \mathcal{V}$ 
2  $V_o = \emptyset$ 
3  $V_s = \emptyset$ 
4 while  $V \neq \emptyset$  do
5     select  $i$  having the smallest  $\bar{d}_i$  in  $V$ 
6     if no  $j, j' \in N_{G_k}[i]$  hold the same component then
7         for all nodes in  $N_{G_k}[i]$  without component, allocate components
            such that no  $j \in N_{G_k}[i]$  and  $j' \in N_{G_k}[i]$  hold the same component
8          $V \leftarrow V \setminus \{i\}, V_o \leftarrow V_o \cup \{i\}$ 
9     else
10         $V \leftarrow V \setminus \{i\}, V_s \leftarrow V_s \cup \{i\}$ 
11 for every  $j \in V_s$  such that  $j$  holds no component do
12    allocate the farthest component  $c$  to  $j$ 
```

in V_o , or in V_s . We know that every node in V_o can access the k components in only one hop. For nodes in V_s , we show the result by contradiction.

Let i be a node in V_s . Assume that i is selected in iteration t , and, after the execution of the algorithm, a component is absent within 3 hops of i . The assumption implies that there is no node in V_o being at two hops from i . Since i belongs to V_s , we know that, at iteration t , there are at least two nodes in $N_{G_k}[i]$, say j and j' , that hold the same component. In the algorithm, the allocation of component during the first phase occurs only when a node is put in V_o . So, j and j' are either in V_o , or neighbors of a node in V_o . Together with the fact that once a node is included in V_o or V_s , it will never be moved out, we conclude that there must exist a node in V_o being less than two hops from i . Hence a contradiction and the lemma follows. \square

Theorem 4. *For any $k \geq 3$, Algorithm 1 gives an integer solution with a total cost that is no more than $\frac{3}{2}k - \frac{5}{2}$ times that of the fractional optimal solution for k -CMSP.*

PROOF. For every node i' in V_o , we know that $d_{i'} = \bar{d}_{i'}$. For a node i in V_s (by the end of phase one), there are two cases: 1) the component at i is assigned in processing another node in phase one, and i 's component coincides with the component held by one of its $k - 1$ nearest sites, 2) two nodes in $N_{G_k}(i)$ hold

the same component. We will treat the two cases separately.

In the first case, the component at i has been assigned in treating some node $i' \in V_o$. We also know that $i \in N_{G_k}(i')$. If i is selected at iteration t , then i' is selected at iteration $t' < t$, so $\bar{d}_{i'} < \bar{d}_i$. Assume that the component at i is 1, and components at $j' \in N_{G_k}(i')$ are $c = \{2, \dots, k\}$. Then the cost of i can be calculated as follows:

$$\begin{aligned} \sum_{j \in F(i)} d_{ij} &\leq \sum_{j' \in N_{G_k}(i')} d_{ij'} = \sum_{c=2}^k d_{ij'_c} \leq \sum_{c=2}^k (d_{ii'} + d_{i'j'_c}) = \\ &(k-1)d_{ii'} + \sum_{c=2}^k d_{i'j'_c} = (k-2)d_{i'i} + \sum_{c=1}^k d_{i'j'_c} \leq \\ &(k-3)\bar{d}_{i'} + \bar{d}_i \leq (k-2)\bar{d}_i \end{aligned}$$

In the second case, from the proof of Lemma 4, we know that there must be a node i' in V_o within 2 hops from i , and the inequality $\bar{d}_{i'} < \bar{d}_i$ holds. Assume that j_1 and j_2 are the two nodes that prevent i from entering V_o , and $d_{ij_1} \leq d_{ij_2}$. Without loss of generality, we can assign component 1 to j_1 . If i is assigned a component in the second phase, then this component can not be component 1, as j_1 is among the nearest neighbors of i . According to the algorithm, we have $j_1 \in N_{G_k}(i) \cap N_{G_k}(i')$. When the placement of components is finished, node j_1 is included in $F(i)$, since i and j_1 hold different components. Then the cost of i can be calculated as follows:

$$\begin{aligned} \sum_{j \in F(i)} d_{ij} &\leq \sum_{j' \in N_{G_k}(i')} d_{ij'} = \sum_{c=1}^{k-1} d_{ij'_c} \leq d_{ij_1} + \sum_{c=2}^{k-1} (d_{i'j'_c} + d_{i'j_1} + d_{ij_1}) = \\ &\sum_{c=1}^{k-1} d_{i'j'_c} + (k-3)d_{i'j_1} + (k-1)d_{ij_1} \leq \\ &\sum_{c=1}^{k-1} d_{i'j'_c} + (k-3)d_{i'j_1} + \frac{k-1}{2}(d_{ij_1} + d_{ij_2}) \leq \\ &(k-2) \sum_{c=1}^{k-1} d_{i'j'_c} + \frac{k-1}{2} \sum_{c=1}^{k-1} d_{ij_c} = \\ &(k-2)\bar{d}_{i'} + \frac{k-1}{2}\bar{d}_i \leq \left(\frac{3}{2}k - \frac{5}{2}\right)\bar{d}_i \end{aligned}$$

As $(k - 2) \leq (\frac{3}{2}k - \frac{5}{2})$ for any k greater than 3, the solution returned by the algorithm has a performance ratio of $\frac{3}{2}k - \frac{5}{2}$ in relation to the fractional optimal solution to k -CMSP. \square

In the execution of the approximation algorithm, every site i needs first determine $N_{G_k}(i)$. Let $n = |\mathcal{V}|$ as before, using the same method proposed in [48], the determination of $N_{G_k}(i), \forall i \in \mathcal{V}$ can be finished in $n \log n$. Then we use merge sort to arrange \bar{d}_i in an ascending order, and the process costs $n \log n$ computations. Finally, $\frac{k(k-1)}{2}$ comparisons is necessary to allocate the component on one site. Therefore, as k is a constant, the complexity of the algorithm is $\mathcal{O}(n \log n)$.

5. Heuristic Algorithms

We describe now three heuristic algorithms. Each of them is motivated by a specific rationale: domination theory in graph, intuition and distributed computation, and fairness consideration. The first two heuristics attempt to minimize the overall cost of k -CMSP. The third heuristic emphasizes on the largest site cost to achieve better fairness. Their respective performances are evaluated in Section 6. The main reason of using heuristic algorithms is that they can produce very fast solutions to large instances of NP-hard problems. Heuristics do not have a performance guarantee. Empirically, however, heuristics sometimes deliver better solutions than an approximation algorithm (see also Section 6). In addition, a heuristic may enable localized computation, which is a useful feature in distributed systems.

5.1. A Heuristic using Domatic Partition

One idea of constructing a heuristic algorithm for k -CMSP is to build a nearest-neighbors graph and compute a domatic partition of it, with the underlying motivation that a domatic partition is composed by a set of dominating sets. If all the nodes in a dominating set are allocated the same component, then the component becomes available to all nodes in one hop. Thus the ideal case is a successful domatic partitioning into k sets on a $k - 1$ -nearest neighbor graph.

Yet, domatic partition of a general graph is NP-complete. However, exact solutions can be computed for several classes of graphs [42]. We focus in this paper on one of these classes, namely the *interval graph*, because a linear-time algorithm is known for computing a domatic partition on any interval graph, and an interval graph is *domatically full*, *i.e.*, k domatic partitions can be found in an interval graph with a minimal degree equal to $k - 1$. The idea is to *augment* a $(k - 1)$ -nearest graph by additional edges, so that it becomes an interval graph. This technique is called *interval completion*. Next, we compute the domatic partition of this interval-completed $(k - 1)$ -nearest graph.

A few recent papers have dealt with domatic partition in ad-hoc networks [43, 41] and peer-to-peer networks [44]. These are works related to our algorithm development, although our objective is different. In these references, a given graph is partitioned into a number of partitions equal to the minimal graph degree, and the solution approaches are based on variants of a well-known randomized approximation algorithm [42]. In our case, we partition a complete weighted graph into k partitions in respect of the weights.

Recall that $G_k = (\mathcal{V}, E_k)$ is the k -nearest graph associated with our distance function, *i.e.* an edge $\{u, v\} \in E_k$ means that either u belongs to the k nearest sites of v , or v belongs to the k nearest sites of u . Determining any domatic partition greater than three for k -nearest graphs in polynomial time is open. Our proposal is to transform G_k into an interval graph. A graph is an interval graph if, to each vertex, we can assign an interval of the real line such that there is an edge between two vertices if and only if their respective intervals have a non-empty intersection. For interval graphs, a domatic partition of size equal to the minimal graph degree plus one can be found in linear time in the size of the graph [45].

A graph is an interval graph if and only if there exists an *interval ordering* of its vertices, *i.e.* a linear ordering $\sigma = (x_1, \dots, x_n)$ of \mathcal{V} such that if there is an edge between x_i and x_k with $i < k$, then for any j such that $i < j \leq k$ there is an edge between x_i and x_j . From a k -nearest graph $G_k = (\mathcal{V}, E_k)$ and an ordering σ of the vertices, we can define an interval graph $G_k^\sigma = (\mathcal{V}, F_k)$, where

for any $1 \leq i < j \leq j' \leq n$, edge (x_i, x_j) belongs to F_k if edge $(x_i, x_{j'})$ belongs to E_k . We remark that all edges of G_k are also present in G_k^σ . Hence G_k^σ is obtained by augmenting G_k .

To use interval graph in our study, design of an interval ordering that results in interval completion and optimizes a given objective is of key importance. Yet, determining the minimal number of edges for an interval completion is NP-hard. In our study, we analyze three known algorithms for interval ordering. The first one, based on a *Lexicographic Breadth First Search* (Lex-BFS) [46], has a linear time complexity. The last vertex of this ordering can be chosen as an extremal one for an interval representation of an interval graph. The second algorithm consists of applying Lex-BFS two times (2-Lex-BFS) because the choice of the starting vertex has a strong impact on the effectiveness of Lex-BFS. Finally, we implement a recent algorithm, denoted K-I [47], which determines a *minimal interval completion*, that is, no subgraph of the generated interval graph is an interval graph. The K-I algorithm runs in $\mathcal{O}(n \cdot m)$ time with m being the number of edges in G_k .

Our algorithm consists in the following steps: (i) build the $k - 1$ nearest graph G_{k-1} , (ii) determine a linear ordering σ of \mathcal{V} , (iii) compute the interval graph G_{k-1}^σ , (iv) compute a domatic partition of G_{k-1}^σ in time $\mathcal{O}(m)$ [45], and (v), if the number of the domatic partition is greater than k , merge the smallest partitions. Note that a k -nearest graph can be computed in $\mathcal{O}(n \log n)$ time [48] and the number of edges m is bounded by $k * n$ in G_k , so the complexity of the entire algorithm is $\mathcal{O}(n \log n)$.

We have compared the performance of the three variants of the domatic heuristic, Lex-BFS, 2-Lex-BFS, and K-I, on a basic simulator where the instances are obtained by uniform distribution of sites in a two-dimensional Euclidean space. Intuitively, a small number of added edges in graph augmentation means that the domatic partition uses preferentially the edges of the k -nearest graph. Therefore the number of added edges is a relevant performance indicator.

In Fig. 1, instances of 250 sites are used to illustrate the number of added edges when the number of partitions k grows from 3 to 10. Not surprisingly, the

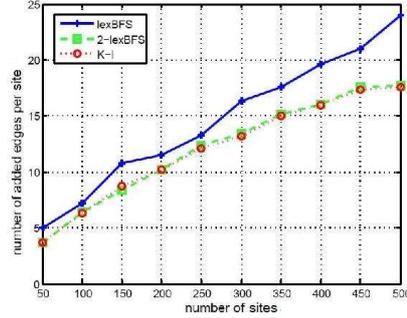
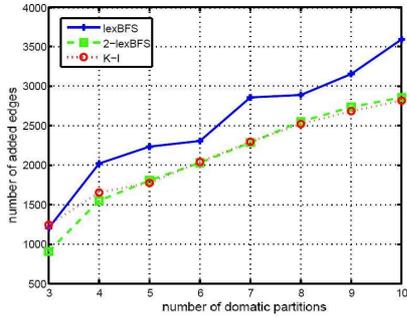


Figure 1: Impact of k on the number of added edges Figure 2: Impact of n on the average number of added edges.

number of added edges increases by graph density. Both 2-Lex-BFS and K-I outperform Lex-BFS algorithm by around 20%. To one's disappointment, the K-I algorithm has no noticeable advantage in performance over the linear-time 2-Lex-BFS algorithm. Fig. 2 shows how the average number of added edges per site for a 6-nearest graph varies over the number of sites. It can be seen that all algorithms produce high numbers of added edges, and this number increases by problem size. Both figures reveal that an interval completion substantially expands the k -nearest graph, and 2-Lex-BFS exhibits, with a linear-time complexity, a performance comparable to that of K-I. We therefore choose to use 2-Lex-BFS in the following experiments.

5.2. An Intuitive and Localized Heuristic

We observe that applying k -nearest graph on undirected edges may lead to poor solutions. See, for example, Figure 3 with 4 components and a 3-nearest graph. Assume that the left part of the graph is firstly allocated components. Vertex i is assigned c_4 . Now we need to decide the component for j . It is easy to see that the overall cost is minimized by assigning c_1 to j , whereas the worst selection is c_4 . If we consider the edges as undirected, node i is currently dominated by all components, and therefore a randomly chosen component will be allocated to j , potentially leading to the worst selection c_4 . However, if we consider the directed graph instead, only the three nearest vertices of i are

considered. In this case i is not dominated by c_1 , which becomes the choice of allocation at j .

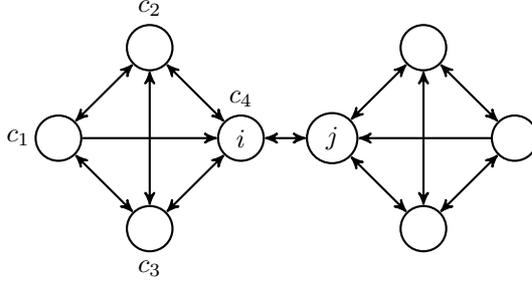


Figure 3: An example of a directed 3-nearest graph.

Before describing the algorithm, we remark that it needs an ordering for traversing all the sites in the network. In order to be consistent with the interval completion approach, we use 2-Lex-BFS to calculate the ordering.

The algorithm is composed by two phases. In the first phase, each site chooses its component based on the components of its one-hop neighbors, and tries to optimize the cost function in component selection. If the components cannot be distinguished by the cost function at a site, the choice becomes pending, and the site is put into a list to be processed later. In the second phase, the sites in the list choose the component allocation to maximize the benefits (*i.e.*, the cost saving) of themselves and their neighbors. We denote by, respectively, $N_{out}(i)$ and $N_{in}(i)$ the k nearest sites of i and the sites that consider i as one of their k nearest sites. Note that a site regards itself to be one of the k nearest sites.

The first phase, performed by an individual site $i \in \mathcal{V}$, is depicted in Algorithm 2. Obviously, site i should avoid choosing a component that is already allocated to any of its nearest sites. The set containing these components is defined in line 1. In addition, site i should not pick a component that has been chosen by any site in $N_{in}(i)$, because these sites have i as a nearest site. Likewise, any component allocated to any nearest site of the sites in $N_{in}(i)$ should be avoided at i . These observations lead to the two sets of components defined

in lines 2–3. If there are components remaining, one of them is randomly selected by i (line 6), otherwise the allocation of i is pending, meaning that the selection will be made in the next phase.

Algorithm 2: Intuitive Heuristic (site i) – first phase

```

1  $AllocC_{out} = \{\varphi(j) : j \in N_{out}(i)\}$ 
2  $AllocC_{in} = \{\varphi(j) : j \in N_{in}(i)\}$ 
3  $AllocC_{inout} = \{\varphi(j') : \exists j \in N_{in}(i), j' \in N_{out}(j)\}$ 
4  $PossibleC = \mathcal{C} \setminus (AllocC_{out} \cup AllocC_{in} \cup AllocC_{inout})$ 
5 if  $|PossibleC| \geq 1$  then
6     randomly choose  $c \in PossibleC$ 

```

Algorithm 3 shows the procedure used to select component at site i that is pending by the end of phase one. The idea is to evaluate the overall gain in terms of cost saving among all possible components. For compactness, we denote by $d(j|c)$ the cost for site j to obtain component c from the closest site currently hosting the component, *i.e.*, $d(j|c) = \min_{\{j \neq i: \varphi(j)=c\}} d(i, j)$. The computation of the gain for site j in $N_{in}(i)$, provided that site i picks component c , is the difference between $d(j|c)$ and $d(j, i)$. It should be remarked that this calculation requires site j to acquire $d(j|c)$. In a distributed system, the calculation will impose some signaling overhead.

Algorithm 3: Intuitive Heuristic (site i) – second phase

```

1 for  $c \in \mathcal{C}$  do
2      $gain(c) = d(i|c) + \sum_{j \in N_{in}(i)} (d(j|c) - d(j, i))$ 
3 pick  $c$  with maximal  $gain(c)$ 

```

Note that each site decides its own component based on information related to sites in two hops from itself. Hence, the algorithm satisfies our requirement that only local information is necessary for each site to select its component. This characteristic fits the characteristics of site-to-site and ad-hoc networks.

Since the heuristic can execute in a distributed manner, we examine the complexity per site. Every site needs to decide its $k - 1$ nearest neighbors, which costs $\mathcal{O}(n \log n)$ comparisons. The complexity of the component alloca-

tion process in the first phase is $\mathcal{O}(k)$. In the second phase, a site i needs to execute a breadth first search (BFS) in two hops to determine the missing component of $N_{in}(i)$ and itself. Then $\mathcal{O}(n)$ computation is needed for i to choose its component. Therefore, the complexity for one site to run the algorithm is $\mathcal{O}(n \log n)$.

5.3. A Heuristic Addressing Fairness

Both previous heuristics exhibit the risk to promote the overall performance over fairness, by sacrificing some sites in order to reduce the total network cost. This may lead to great difference in the costs of sites. In a practical system, this unfairness has the risk to frustrate the users connected to the most poorly served sites. The heuristic we describe now looks for solutions based on the principle of max-min fairness, that is, we try to minimize the network cost of the most disadvantaged site with priority. Another motivation for this heuristic comes from the observation that a site being located close to $k - 1$ other sites is probably close to many additional sites, so it is not affected by a sub-optimal component allocation in its surroundings, whereas the site that is far from other sites is more sensitive to the location decision.

The algorithm goes through n rounds. In each round, it determines the site that has not been allocated any component and is potentially the most disadvantaged one, measured in its distances to the $k - 1$ nearest sites that may serve it. Formally, for a site v , we note S_v^k the set of sites having the potential to be selected by v for service delivery, and $d(S_v^k)$ the cost to access the service. This set should obviously contain k elements including v itself, that is, $|S_v^k| = k$ and $v \in S_v^k$. At the beginning of the round, any site $u \in S_v^k$ satisfies the condition that either $\varphi(u) = NULL$ or, if $\varphi(u)$ is not null, then there is no site $w \in S_v^k$ with $\varphi(w) = \varphi(u)$.

Once a site, say v , has been selected, we assign distinct components to all sites in S_v^k . Therefore, at the end of the round, sites in S_v^k have components allocated. The components at $S_v^k \setminus \{v\}$ are all different from each other. By considering preferentially the sites that seem to be the most disadvantaged, the

algorithm tries to promote fairness in the allocation.

In Algorithm 4 we describe the procedure of ranking and allocation. At line 4, the algorithm calculates the potential cost of all the sites not having component allocated. At line 5, the algorithm chooses the site v_{max} with the highest potential cost. We determine the components which have not appeared in the neighbor set of v_{max} at line 8, then these missing components are allocated to the free sites in the neighbor set at line 10.

Algorithm 4: Fairness Heuristic

```

1  set  $V \leftarrow \mathcal{V}$ 
2  while  $V \neq \emptyset$  do
3    for  $v \in V$  do
4      determine  $S_v^k$ 
5      determine  $v_{max} = \operatorname{argmax}_{v \in 1, \dots, n} d(S_v^k)$ 
6       $C = \mathcal{C} \setminus \{c : c = \varphi(u), u \in S_{v_{max}}^k\}$ 
7       $V' = \{u | u \in S_{v_{max}}^k \cap \varphi(u) = \text{NULL}\}$ 
8      for  $c \in C$  do
9        randomly choose  $u \in V'$ 
10       set  $\varphi(u) = c$ 
11        $V' \leftarrow V' \setminus \{u\}$ 
12       $V \leftarrow V \setminus \{v_{max}\}$ 

```

A drawback of this algorithm is that it requires full knowledge of the network. However, a multi-site service provider typically has accurate information of the locations of the servers and can hence supply the algorithm with this input. This algorithm has the same complexity as the approximation algorithm $\mathcal{O}(n \log n)$.

6. Simulations

Many recent works, including some in standards organizations, have dealt with matching overlay networks and the Internet. In our simulation, we use as input the measurement results from one of these works. The underlying network is a matrix of latencies between all pairs of 2,500 nodes from the Meridian project³. For each run, we choose randomly n nodes, then, for each pair of

³Measurements have been done in May 2004. For more information, see <http://www.cs.cornell.edu/People/egs/meridian/>

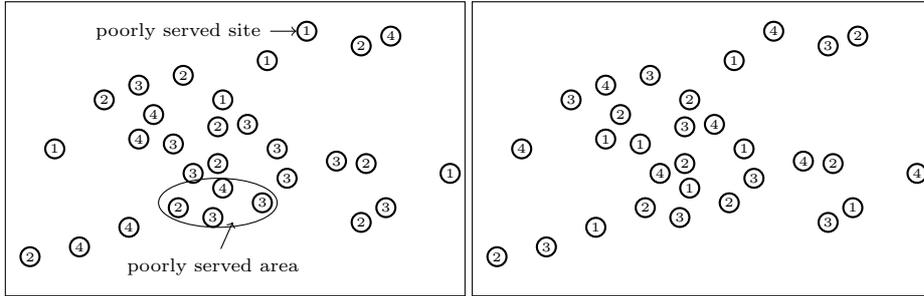


Figure 4: Random allocation.

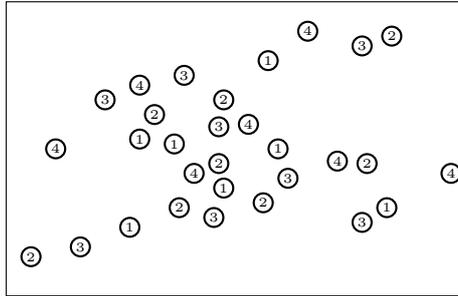


Figure 5: Intuitive heuristic.

nodes, the network cost between them is the actual round-trip time (RTT) that has been measured. Motivated by the importance of delivery time, latency is a typical metric used by service provider to characterize performance. Moreover, it is one of the few metrics that a service provider can easily measure. As this paper addresses the point of view of service provider, we adopt this metric for this set of simulations. Note however that other metrics could have been chosen, for example cross-domain traffic that is of relevance to network operators [7].

Previous works related to multi-site distributed services have suggested to use random strategies in allocating components to sites [6, 10]. This strategy is indeed easy to implement. In addition to comparing our algorithms among themselves, we will illustrate the gain of optimization in comparison to random allocation. Therefore, we display all results under the same form: we compute the average RTT cost for all sites in the allocation given by an algorithm, then we compare it to random allocation, of which the cost is normalized to 1.0. Hence, when a point is close to 1.0, it means that the gain of implementing the algorithm for this parameter setting is negligible. To reduce random effects, more than 20 different instances are tested for each comparison. In addition to the algorithms presented in the previous sections, we include the $3/2k - 1$ approximation algorithm, originally developed for k-PUFLP, in the simulations.

Before showing the simulation results, we first give a small example to gain understanding of allocation solutions and highlight the benefit of our algorithms. In Fig. 4 and Fig. 5, 30 nodes are chosen from the Meridian matrix and displayed

geographically. We decompose an application into 4 components and assign them to these sites. The number on each site represents the component that the site holds. A random allocation is shown in Fig. 4. In the figure, we find poorly served sites. For instance, the highlighted poorly served site is far from component 3, and the set of poorly served sites in the ellipse has long distance to component 1. Fig. 5 shows the location solution by the intuitive heuristic. It is apparent that the sites are closer to all the components in comparison to the random allocation.

6.1. Small Instances

We apply the Gurobi solver [49] to the integer programming model to shed light on the performance of the algorithms in respect of global optimum. In our computations, the time limit for calculating the result for each instance is 5 hours. As k -CMSP is NP-hard, the exponential-amount nature of the computation in solving the model means that exact solution is within reach for relatively small instances only. Nevertheless, the results give some indication on how far from optimality our algorithms and the random allocation scheme perform.

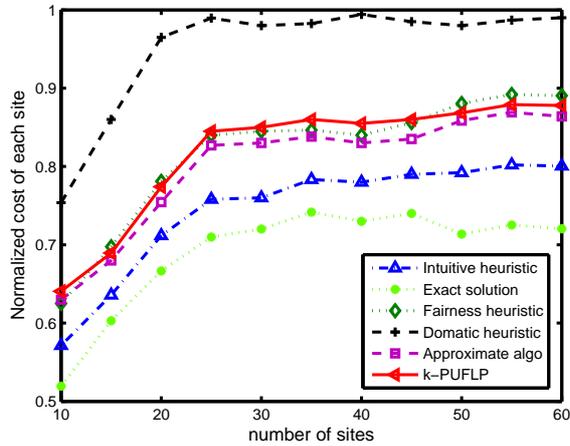


Figure 6: Performance evaluation for small instances: Impact of n on the average cost.

Instance size	45 sites	50 sites	55 sites	60 sites	100 sites
Gap	3.96%	4.19%	4.39%	4.73%	11.32%

Table 1: Average optimality gap.

We show in Fig. 6 curves representing the normalized average costs for various numbers of sites. The number of components is fixed to be 6. This setting is typical for a large-scale service provider managing a set of data-centers. Note that for the instances with more than 45 sites, the Gurobi solver cannot reach the optimal solution in the time limit. The average optimality gap is reported in Table 1. We have also applied Gurobi to instances with 100 sites. However, the large gap indicates that the results are not useful for comparison.

For a small number of sites, all algorithms perform far better than the random one. It is possible to halve the average latency. As the number of sites increases, the improvement becomes less but remains considerable. In particular, the exact solution performs a quarter better than the random one. This, together with the fact that random allocation is currently a common approach in engineering practice, highlight the potential of gains by optimization. Except for the domatic heuristic, the performances of the algorithms seem to stabilize with an approximate gain of 20% over random allocation for $n \geq 25$.

A clear hierarchy of the algorithms' performance is revealed. As can be expected, no algorithm can perform as well as the exact one. We observe however that the intuitive heuristic exhibits performance that is remarkably close to optimum. On the contrary, the domatic heuristic is especially disappointing. Although this algorithm relies on theoretical concepts and results, it performs only slightly better than random allocation. Even for small instances, the number of added edges is so large that the allocation does not significantly leverage on k -nearest graph. The fairness heuristic and the approximate algorithm have almost identical performances. Moreover, our approximate algorithm performs slightly better than the algorithm for k -PUFLP because of the improvement in the approximation ratio.

In Fig. 7, the number of sites is fixed to be 40, and the number of components

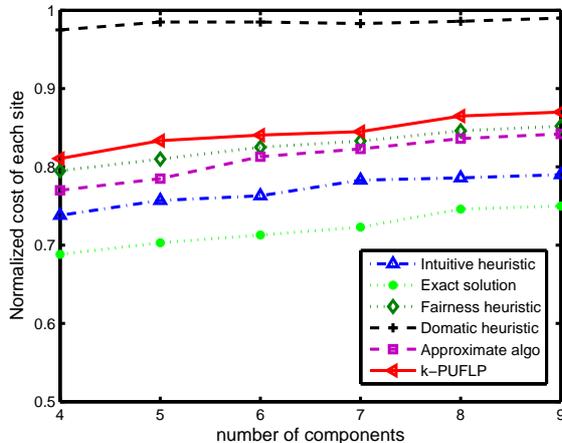


Figure 7: Performance evaluation for small instances: Impact of k on the average cost.

varies. Besides the theoretical interest of measuring the impact of the number of components on algorithm performance, this scenario occurs typically for virtual games where several servers have to synchronize regularly.

The gain that we can see from the algorithms tends to slightly decrease when the number of components increases. This trend can be explained by the fact that the relative distance difference between the p th and the $(p + 1)$ th closest neighbors of a site tends to diminish when p increases. Indeed, consider concentric rings around a site. The area of a ring grows by its radius, thus the number of neighbors belonging to a large ring is bigger than that of a small one when the density is approximately constant. As we compute the average RTT, the benefits from a clever algorithm tends to decrease because there are many closely located neighboring sites.

The major result we extract from both Figures 6 and 7 is that our intuitive heuristic performs remarkably well. Indeed, it gives results that are almost equal to the optimal values. Hence the performance of this heuristic deserves future investigations in various application contexts. On the other hand, the fairness heuristic, aimed at reaching fairness among the sites, exhibits also good performance in the overall cost. It performs close to the approximation algorithm

that has a performance guarantee for any metric distance function.

6.2. Large Instances

We compare now the algorithms to the random allocation scheme on larger instances. The main point we would like to show here is scalability, *i.e.*, whether and how the relative performance of the algorithms change in the numbers of sites and components.

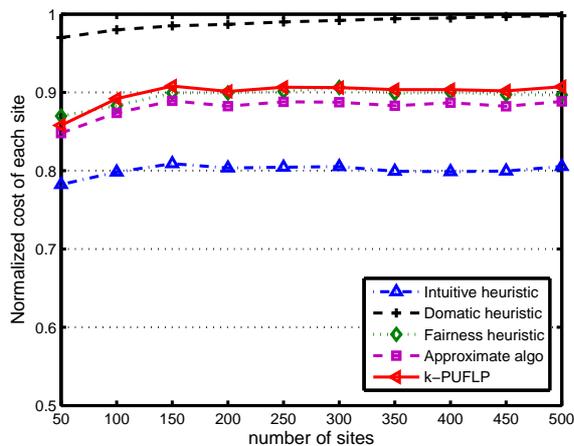


Figure 8: Impact of n on the average cost for large instances.

In Fig. 8, the number of sites grows while the number of components is fixed to 6. The results are consistent to what we have obtained for small instances. When the number of sites becomes large, the domatic heuristic performs almost as poorly as the random allocation scheme. Indeed, the augmented graph contains so many added edges that it looks like a random graph. On the contrary, none of the intuitive heuristic or the fairness heuristic suffers from any scaling effect. The former generates solutions that are 20% better than the random allocation. For a service provider, such an amount of gain in the delivery time is noteworthy.

In Fig. 9, the number of components varies, while the number of sites is fixed to 250. Again, the figure verifies most of the observations made earlier. Note

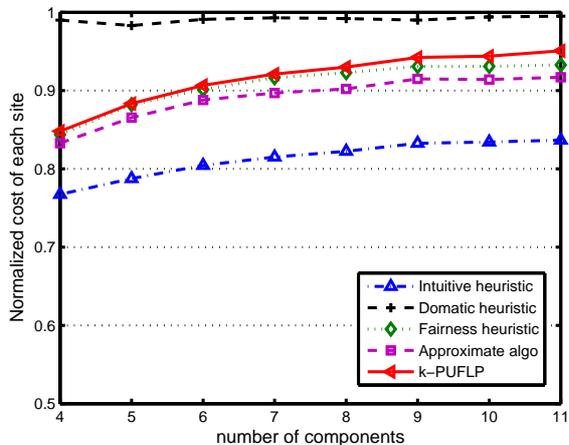


Figure 9: Impact of k on the average cost for large instances.

that the figure indicates, in addition to algorithm performance, the trade-off between two aspects in service distribution: the storage size of the sites, which directly links to the number of components, and the cost of collecting all components for delivering service. Here we emphasize that, for a few components, the gain is apparent, whereas it becomes less impressive for a service with many components. In this latter case, the random allocation, which is easier to implement, is preferable. In other words, it's not likely that our algorithms find their key applications in the case of boxes hosting small portions of a video.

6.3. Variance

Now we show the variance of cost for large instances. For clarity, we do not present the curves of all the algorithms, but the variance of the random allocation, the intuitive heuristic, and the fairness heuristic.

We find from Fig. 10 that the ratio of the maximum cost of sites over the average one increases gradually. The ratio changes from 1.8 to 2.2 for the random allocation, and from 1.75 to 2 for the intuitive heuristic. As it is expected, our fairness heuristic gives the best performance in terms of small variance. The ratio for the fairness heuristic varies from 1.45 to 1.6. The result shows that

both the intuitive heuristic and the fairness heuristic outperform the random allocation. We further analyze the benefit in Section 6.4. In fact, the ratio like 1.75 or 2 is acceptable for certain services as google search engine, since the service provider does not pay much attention on special cases, but the average response time of the engine. And the ratio about 1.5 is reasonable for more applications. Fig. 11 reveals the same results as those of Fig. 10, when the number of sites is fixed and the number of components varies.

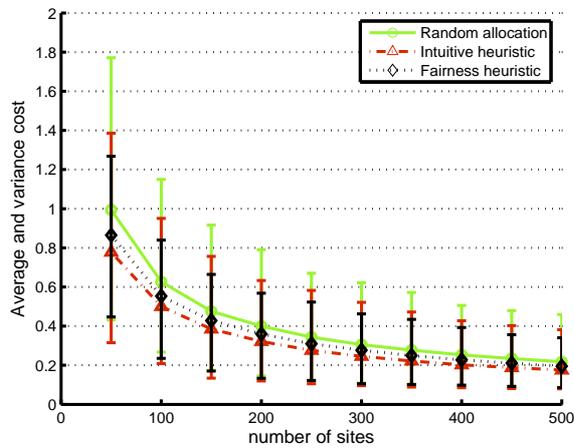


Figure 10: Impact of n on the average and variance cost for large instances.

Moreover, in Fig. 10, the cost decreases with the augmentation of site number since the density of sites increases and the distance between two sites becomes less. On the other side, the cost increases with the increment of component number in Fig. 11 because each site has more components to be accessed.

6.4. Fairness

Although the intuitive heuristic gives good performances in the overall cost, we observe that some sites are not fairly treated. In Fig. 12 and Fig. 13 we investigate the maximum cost generated by different algorithms. That is, for each algorithm and each configuration, we pick up the site with the maximum cost. Then we normalize its cost by the maximum cost produced by the random

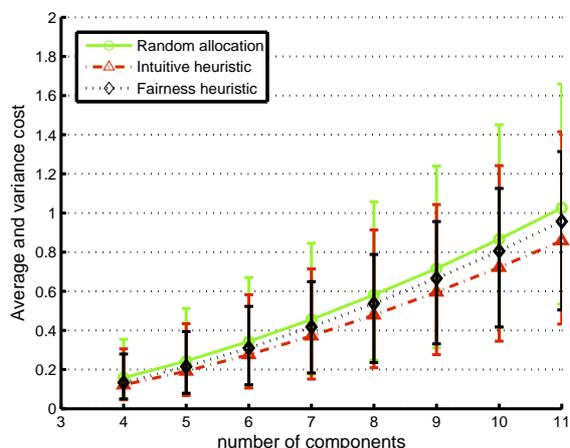


Figure 11: Impact of k on the average and variance cost for large instances.

allocation.

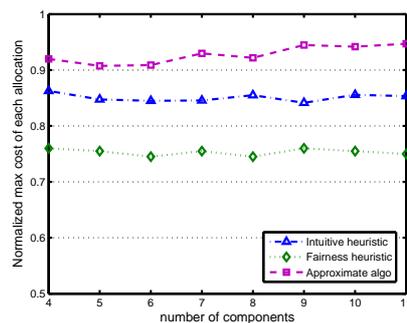
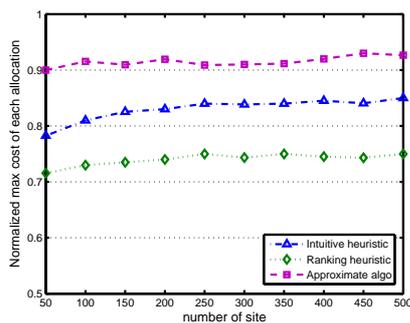


Figure 12: Impact of n on the maximum cost. Figure 13: Impact of k on the maximum cost.

We observe that, as can be expected, the largest cost among sites is better in the allocation by the fairness heuristic. In comparison to the random allocation, the gain is around a quarter. At the mean time, the benefit of the intuitive algorithm is around 15%. However, the performance of the approximate algorithm is close to the random allocation.

The relative performance between the algorithms in Fig. 12 are similar to what has been observed earlier. In Fig. 13, the random allocation does not

perform better when the number of components increases. We note that, in the instances, typically some nodes have larger distances values in their k -nearest neighbors than others. For these nodes, allocation of components should be made with priority to the nearest neighbors. Thus the benefit of optimization in the fairness metric remains when k grows.

In Fig. 14 and Fig. 15, we examine the standard deviation given by different algorithms. The same as before, we normalize the standard deviation of our algorithms by the standard deviation produced by the random allocation. Both figures indicate that the gain of the fairness heuristic is significant (around 40%). The intuitive and approximate algorithm also outperform the random allocation.

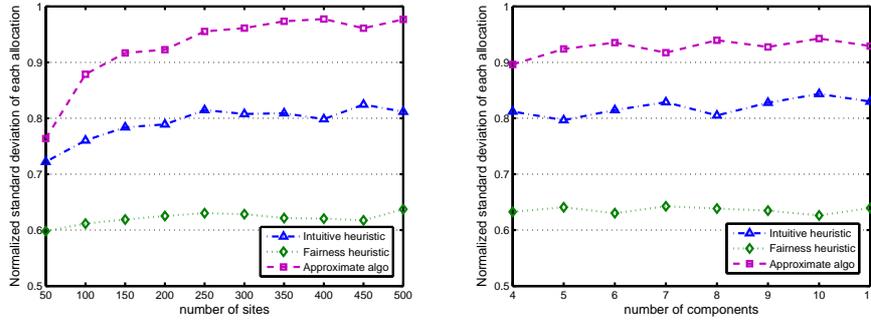


Figure 14: Impact of n on the fairness metric. Figure 15: Impact of k on the fairness metric.

Unlike the fairness algorithm, whose performance is stable, the benefits of the intuitive and approximate algorithm decrease with the increment of sites. The gain of the intuitive heuristic stabilizes at 20%, while the performance of approximate algorithm becomes similar with the random allocation since the approximate algorithm give the priority to the sites with small cost. The Fig. 15 shows that the benefits of our algorithms do not vary with k , which is consistent with the result of Fig. 13.

7. Summary and Conclusions

We have studied theoretical and algorithmic aspects of an optimal network locality problem in the application context of locating and delivering distributed services in a multi-site cloud architecture. The main characteristic of the problem is the distribution of a (possibly large) number of distinct service components. The objective of the optimization is the cost for service delivery. In addition to studying problem complexity, we have presented an integer programming model for the purpose of performance evaluation. One approximation algorithm and three heuristic algorithms are developed and studied numerically. Among them, the intuitive heuristic performs close to optimality for the tested networks where optimum can be computed via integer programming. For large networks this heuristic yields significantly better solutions than the random allocation. Moreover, the fairness heuristic achieves a good balance between the overall cost and the maximum one over sites.

There are several potential extensions of the work. One is the investigation of the impact of distance functions, the distribution of sites, and topology structures in various types of networks on the location solution. In view of the NP-completeness of the problem class, one issue is the design of approximation algorithms with better performance ratio, and the identification of classes of cost functions where exact solutions are within reach in polynomial time. Another extension is the inclusion of constraints modeling the sites' ability of serving each other in terms of the upstream communication capacity or the computational capacity, in order to avoid overloading. Finally, allocating multiple components on one site and fault tolerance are also interesting extensions.

Acknowledgment

We would like to thank Lei Chen, Department of Science and Technology, Linköping University, for providing us with the computational results of solving the integer model of k -CMSP.

- [1] L. A. Barroso and U. Hölzle, “The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines,” *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.
- [2] Amazon elastic computing cloud (amazon ec2), <http://aws.amazon.com/ec2/>.
- [3] Azure services platform, <http://www.microsoft.com/azure/default.mspix>.
- [4] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, “The little engine(s) that could: scaling online social networks,” in *Proceedings of the ACM SIGCOMM*, 2010, pp. 375–386.
- [5] K. Kant, “Data center evolution:: A tutorial on state of the art, issues, and challenges,” *Computer Networks*, vol. 53, no. 17, pp. 2939–2965, 2009.
- [6] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, “Greening the Internet with Nano Data Centers,” in *Proc. of ACM CoNEXT*, 2009.
- [7] Y. Chen, J. Leblet, and G. Simon, “On Reducing the Cross-Domain Traffic of Box-Powered CDN,” in *ICCCN: 18th IEEE International Conference On Computer Communications and Networks*, August 2009.
- [8] J. Pujol, V. Erramilli, and P. Rodriguez, “Divide and Conquer: Partitioning Online Social Networks,” *Arxiv preprint arXiv:0905.4918*, 2009.
- [9] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Plachouras, and L. Telloi, “On the feasibility of multi-site web search engines,” in *Proc. of ACM Conf. on Information and knowledge management*, 2009, pp. 425–434.
- [10] A. Nafaa, S. Murphy, and L. Murphy, “Analysis of a Large-Scale VOD Architecture for Broadband Operators: A P2P-Based Solution,” *IEEE Communications Magazine*, December 2008.
- [11] K. Church, A. Greenberg, and J. Hamilton, “On delivering embarassingly distributed cloud services,” *Hotnets VII*, 2008.

- [12] S. Paul, R. Yates, D. Raychaudhuri, and J. Kurose, “The cache-and-forward network architecture for efficient mobile content delivery services in the future Internet,” in *Proc. of ITU-T Kaleidoscope Academic Conference on Innovations in NGN: Future Network and Services*, 2008.
- [13] J. Rothschild, “High performance at massive scale - lessons learned at facebook.” <http://cns.ucsd.edu/lecturearchive09.shtml>, Oct. 2009.
- [14] B. Li and R. Li, “Approximation algorithm for k-puffpn,” *Engineering and Applications (in Chinese)*, vol. 44, pp. 97–99, 2008.
- [15] D. B. Shmoys, E. Tardos, and K. I. Aardal, “Approximation algorithms for facility location problems,” in *Proc of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 265–274.
- [16] F. A. Chudak and D. Shmoys, “Improved approximation algorithms for the uncapacitated facility location problem.” 1998, unpublished manuscript.
- [17] K. Jain, M. Mahdian, and A. Saberi, “A new greedy approach for facility location problems,” in *ACM Symposium on Theory of Computing (STOC)*, 2002.
- [18] M. Mahdian, Y. Ye, and J. Zhang, “Improved approximation algorithms for metric facility location problems,” in *5th Int. Workshop on Approximation Algorithms for Combinatorial Optimization*, vol. 2462, 2002, pp. 229–242.
- [19] S. Guha and S. Khuller, “Greedy strikes back: Improved facility location algorithms,” *Journal of Algorithmm*, vol. 31, pp. 228–248, 1999.
- [20] S. H. Owen and M. S. Daskin, “Strategic facility location: A review,” *Euro. Journal of Operational Research.*, vol. 111, pp. 423–447, 1998.
- [21] J. G. Klincewicz, H. Luss, and E. Rosenberg, “Optimal and heuristic algorithms for multiproduct uncapacitated facility location,” *European Journal of Operational Research*, vol. 26, pp. 251–258, 1986.

- [22] J. G. Klincewicz and H. Luss, “A dual based algorithm for multiproduct uncapacitated facility location,” *Transportation Science*, vol. 21, pp. 198–206, 1987.
- [23] H.-C. Huang and R. Li, “A k-product uncapacitated facility location problem,” *European Journal of Operational Research*, vol. 185, no. 2, pp. 552 – 562, 2008.
- [24] M. Charikar, S. Guha, D. B. Shmoys, and E. Tardos, “A constant factor approximation algorithm for the k -median problem,” in *Proc. of ACM STOC '99*, 1999, pp. 1–10.
- [25] K. Jain and V. V. Vazirani, “Primal-dual approximation algorithms for metric facility location and k-median problems,” in *Proc. of IEEE FOCS'99*, 1999.
- [26] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, “Analysis of a local search heuristic for facility location problems,” in *Proc. of ACM-SIAM SODA '98*, 1998, pp. 1–10.
- [27] V. Arya, N. Garg, R. Khandekar, and V. Pandit, “Local search heuristic for k-median and facility location problems,” in *Proc. of ACM-SIAM SODA '98*, 1998, pp. 1–10.
- [28] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, and A. Bestavros, “Distributed placement of service facilities in large-scale networks,” in *Proc. of IEEE INFOCOM*, 2007.
- [29] Özgür Ulusoy, “Research issues in peer-to-peer data management,” in *Proc. 22nd international symposium on Computer and Information Sciences*, 2007.
- [30] S. Goel and R. Buyya, “Data replication strategies in wide area distributed systems,” *Enterprise Service Computing: From Concept to Deployment*, 2006.

- [31] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and replication in unstructured peer-to-peer networks,” in *Proc. 16th International Conference of Supercomputing*, 2002.
- [32] S. Ata, Y. Gotoh, and M. Murata, “Replication strategies in peer-to-peer services over power-law overlay networks,” in *Proc. of the 7th Asia-Pacific Network Operations and Management Symposium*, 2003.
- [33] E. Leontiadis, V. V. Dimakopoulos, and E. Pitoura, “Creating and maintaining replicas in unstructured peer-to-peer systems,” *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, 2006.
- [34] A. Oram, “Peer-to-peer: harnessing the power of disruptive technologies,” *O’Reilly*, 2001.
- [35] S. Rajasekhar, B. Rong, K. Y. Lai, I. Khalil, and Z. Tari, “Load sharing in peer-to-peer networks using dynamic replication,” in *proc. of the 20th International Conference on Advanced Information Networking and Applications*, vol. 1, 2006.
- [36] K. Ranganathan, A. Iamnitchi, and I. Foster, “Improving data availability through dynamic model-driven replication in large peer-to-peer communities,” in *Proc. of 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.
- [37] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of web server replicas,” in *In Proceedings of IEEE INFOCOM 2001*, 2001.
- [38] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, “On the placement of internet instrumentation,” in *In Proceedings of IEEE INFOCOM 2000*, 2000.
- [39] G. Pallis and A. Vakali, “Insight and perspectives for content delivery networks,” *Communications of the ACM*, vol. 49, no. 1, pp. 101–106, 2006.

- [40] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co., 1979.
- [41] S. V. Pemmaraju and I. A. Pirwani, “Energy conservation via domatic partitions,” in *Proc. of ACM Int. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2006, pp. 143–154.
- [42] U. Feige, M. M. Halldórsson, G. Kortsarz, and A. Srinivasan, “Approximating the domatic number,” *SIAM J. Comput.*, vol. 32, no. 1, pp. 172–195, 2002.
- [43] T. Moscibroda and R. Wattenhofer, “Maximizing the Lifetime of Dominating Sets,” in *Proc. of IEEE Int Parallel and Distributed Processing Symposium (IPDPS)*, 2005, pp. 242b–242b.
- [44] G. Dan, “Cooperative caching and relaying strategies for peer-to-peer content delivery,” in *Proc. of Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.
- [45] A. S. Rao and C. P. Rangan, “Linear algorithm for domatic number problem on interval graphs,” *Information Processing Letters*, vol. 33, no. 1, pp. 29 – 33, 1989.
- [46] M. Habib, R. M. McConnell, C. Paul, and L. Viennot, “Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing,” *Theor. Comput. Sci.*, vol. 234, no. 1-2, pp. 59–84, 2000.
- [47] K. Suchan and I. Todinca, “Minimal interval completion through graph exploration,” *Theor. Comput. Sci.*, vol. 410, no. 1, pp. 35–43, 2009.
- [48] D. Eppstein, M. S. Paterson, and F. F. Yao, “On nearest neighbor graphs,” *Discrete & Computational Geometry*, vol. 17, no. 3, pp. 263–282, April 1997.
- [49] Gurobi Optimizer, <http://www.gurobi.com/>.