On Optimal Worst-Case Matching

Cheng Long[†] Raymond Chi-Wing Wong[†] Philip S. Yu[‡] Minhao Jiang[†]
[†]The Hong Kong University of Science and Technology, [‡]University of Illinois at Chicago
[†]{clong,raywong,mjiangac}@cse.ust.hk, [‡]psyu@cs.uic.edu

ABSTRACT

Bichromatic reverse nearest neighbor (BRNN) queries have been studied extensively in the literature of spatial databases. Given a set P of service-providers and a set O of customers, a BRNN query is to find which customers in O are "interested" in a given service-provider in P. Recently, it has been found that this kind of queries lacks the consideration of the *capacities* of service-providers and the *demands* of customers. In order to address this issue, some spatial matching problems have been proposed, which, however, cannot be used for some real-life applications like emergency facility allocation where the maximum matching cost (or distance) should be minimized. In this paper, we propose a new problem called *SPatial Matching for Minimizing Maximum matching distance* (SPM-MM). Then, we design two algorithms for SPM-MM, *Threshold-Adapt* and *Swap-Chain*. *Threshold-Adapt* is simple and easy to understand but not scalable to large datasets due to its relatively high time/space complexity. *Swap-Chain*, which follows a fundamentally different idea from *Threshold-Adapt*, runs faster than *Threshold-Adapt* by orders of magnitude and uses significantly less memory. We conducted extensive empirical studies which verified the efficiency and scalability of *Swap-Chain*.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms

Algorithms, Experimentation, Performance

Keywords

Optimal worst-case spatial matching, Bottleneck matching

1. INTRODUCTION

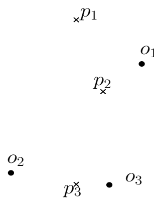
Bichromatic reverse nearest neighbor (BRNN) queries have been studied extensively [17, 18, 24]. Let P be a set of service-providers and O be a set of customers. A BRNN query is to find which customers in O are "interested" in a given service-provider in P. However, BRNN queries lack the consideration of the *capacities* of service-providers and the *demands* of customers. In order to address this issue, some *spatial matching problems* [25, 22, 21] have been proposed which assign service-providers to customers with the above consideration.

In some real-life applications like hospital allocation, a common goal is to minimize the *maximum* distance (or cost) between a hospital and a residential estate served by this hospital. For example, in the Hong Kong ambulance service, the minimized maximum cost is about 12 minutes (driving distance) [1].

To illustrate, we go through a toy example as shown in Figure 1. In Figure 1(a), P contains three hospitals p_1, p_2 and p_3 and O contains three residential estates o_1, o_2 and o_3. Figure 1(b) shows all pairwise distances between P and O. For the sake of illustration, suppose that the *capacity* of each hospital p in P is 1, which means that the greatest amount of the service given by p is 1, and the *demand* of each residential estate o in O is also 1, which means that the amount of the service requested by o is 1. In this case, each hospital can serve at most one residential estate. In order to minimize the maximum distance between a hospital and the residential estate served by this hospital, we form an *assignment* between P and O as shown in Figure 2(a). In this assignment, p_1, p_2 and p_3 serve o_1, o_3 and o_2, respectively. If p serves o, we draw a line between p and o in the figure. The number next to the line is called the *matching distance* between p and o which corresponds to the Euclidean distance between p and o. In this assignment, the *maximum matching distance* (*mmd*) is equal to 6. Besides, we cannot find any other assignment which satisfies the service demand of each customer and has its *mmd* smaller than 6. Thus, 6 is the *optimal mmd*.

In this paper, we propose a new problem called *SPatial Matching for Minimizing Maximum matching distance* (SPM-MM). Given a set P of service-providers each of which has a capacity and a set O of customers each of which has a demand, the SPM-MM problem is to assign the service-providers in P to the customers in O with the consideration of the capacities of the service-providers such that the demand of each customer in O is satisfied and the maximum matching distance (i.e. *mmd*) is minimized.

SPM-MM has extensive applications in matching between two sets of objects where the *worst-case* cost should be minimized. The notions of "service-provider" and "customer" in SPM-MM are general and can have alternative semantics in different (even non-geographic) applications. One such application is the allocation problem between emergency facilities and users. Hospitals, fire stations and police stations are some examples of emergency facilities and residential estates and commercial areas are some examples of users. Logistics, data warehouse allocation and mail delivery are some applications with non-emergency facilities. *Profile matching* [25] is another application where we want to match "items" (re-

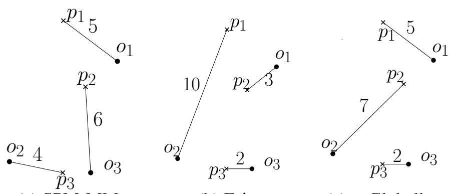


(a) Spatial layout

dist	o_1	o_2	o_3
p_1	5	10	11
p_2	3	7	6
p_3	9	4	2

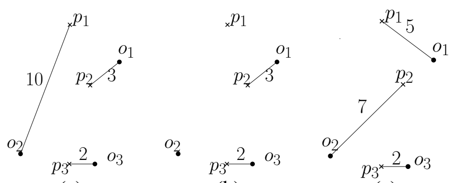
(b) Pairwise distances

Figure 1: A running example



(a) SPM-MM (b) Fair (c) Globally-optimized

Figure 2: Spatial matching problems



(a) (b) (c)

Figure 3: The process of *Swap-Chain*

garded as service-providers) with "customers" such that the worst-case dissatisfactory rate among all customers is minimized.

It turns out that SPM-MM reduces to be a classical problem in computer science, *Bottleneck Matching Problem* (BMP) [14], when each service-provider has its capacity equal to 1 and each customer has its demand equal to 1 as well. Given two sets of n objects, A and B, and the cost of matching each object in A with each object in B, the BMP problem is to find the *perfect matching* with the smallest *cost* among all perfect matchings between A and B where the *cost* of a perfect matching M is defined to be the greatest cost of matching an object from A and an object from B in M. It can be verified that SPM-MM becomes BMP when $|P| = |O|$, each service-provider $p \in P$ (customer $o \in O$) has its capacity (demand) equal to 1, and the distance between p and o is used as the cost of matching p with o for each $p \in P$ and each $o \in O$. [4] provides a comprehensive study on existing solutions of BMP, among which, the *Threshold* algorithm is the fastest.

No existing algorithms can be used to solve the SPM-MM problem. Firstly, the algorithms for BMP cannot be used *directly* for SPM-MM since in SPM-MM, the capacities/demands could be arbitrary positive integers. Besides, we will show that an *adapted* version of the *Threshold* algorithm, which is originally designed for BMP, is not scalable for SPM-MM. Secondly, the solutions for all existing spatial matching problems cannot be used for SPM-MM. To illustrate this, we first give a brief background of these problems. Two major types of spatial matching problems have been studied. The first one [25] aims to find the *fair* assignment between P and O which is to assign to each customer the nearest service-provider that has not been exhausted of serving other *closer* customers. Figure 2(b) shows the fair assignment between P and O whose *mmd* is equal to 10 (> 6). The second one is to find the *globally optimized* assignment between P and O which guarantees that each customer's service demand is satisfied and the *overall* matching cost is minimized. Figure 2(c) shows the globally optimized assignment between P and O whose *mmd* is equal to 7 (> 6).

In this paper, we design two algorithms for the SPM-MM problem. The first one is called *Threshold-Adapt* and the second one is called *Swap-Chain*. *Threshold-Adapt* is an algorithm which shares a similar idea as *Threshold* which is originally designed for BMP. Unfortunately, *Threshold-Adapt* is not scalable to large datasets due to its high time/space complexity. *Swap-Chain* is an algorithm which is scalable and runs faster than *Threshold-Adapt* by orders of magnitude by using the concept of finding a series of elements where every two adjacent elements are "close" to each other for re-matching. The operation of finding a "close" element from another element can be implemented efficiently by spatial queries.

It is worth mentioning that our proposed algorithms are not limited to the Euclidean space. In fact, they can also be adapted to non-metric space (with a certain sacrifice of efficiency). For example, our algorithms can be adapted to settle the SPM-MM problem in Figure 1, even if the distance between a hospital and a residential estate is their road-network distance. The discussion on how to adapt our techniques to non-metric space is given in Section 6.

We summarize our main contributions as follows. Firstly, to the best of our knowledge, we are the first to propose the SPM-MM problem, which has extensive real-life applications. Secondly, to solve SPM-MM, we design our first algorithm, *Threshold-Adapt*, based on an idea of one popular solution of BMP, *Threshold*. *Threshold-Adapt* is not scalable for large datasets due to its high time/space complexity. Therefore, we develop another novel algorithm, *Swap-Chain*, which runs faster than *Threshold-Adapt* by orders of magnitude and is scalable to very large datasets (in millions). Finally, we conducted extensive empirical studies on these two solutions.

In the following, Section 2 defines the SPM-MM problem, and Section 3 provides the related work of SPM-MM. Section 4 and Section 5 introduce two algorithms, *Threshold-Adapt* and *Swap-Chain*, respectively. Section 6 gives some discussions. Section 7 includes the empirical studies and Section 8 concludes the paper.

2. THE SPM-MM PROBLEM

Let P be a set of service-providers and O be a set of customers. Each service-provider p (customer o) has a service capacity (demand), denoted by $p.w$ ($o.w$). We represent the Euclidean distance between o and p with $d(o, p)$.

Let $W_O = \sum_{o \in O} o.w$ and $W_P = \sum_{p \in P} p.w$. We assume that the service demands of all customers in O can be satisfied by the service-providers in P, i.e. $W_P \geq W_O$. Under this assumption, it is possible that some service-providers are not matched with customers. In case that $W_P < W_O$, we swap the roles of P and O and thus this assumption still holds.

PROBLEM 1 (SPM-MM). *SPM-MM generates the assignment A denoting a set containing the elements in the form of triplets (o, p, w), where (o, p, w) is called a* match *between o and p and denotes that p provides the service with the amount of w to o. Furthermore, the following three conditions hold.*

- ***Capacity Constraint:*** *No service-provider provides its service of the amount greater than its capacity, i.e., $\forall p \in P$, $\sum_{(o,p,w) \in A} w \leq p.w$.*
- ***Demand Constraint:*** *Each customer's service demand is satisfied, i.e., $\forall o \in O$, $\sum_{(o,p,w)} w = o.w$.*
- ***Optimality Constraint:*** *The* mmd *of A is minimized, i.e., $\max\{d(o, p)|(o, p, w) \in A\}$ is minimized.*

Note that in the following, for clarity, the match (o, p, w) is simply denoted as (o, p) when $w = 1$. □

In order to ease our discussion, we say that an assignment is *full* if it satisfies the Capacity Constraint and the Demand Constraint defined above. Note that there are an exponential number of full assignments. To illustrate, consider the case where $|P| = |O| = n$ and the capacity (demand) of each service-provider (customer) is equal to 1. In this case, there exist $n!$ possible full assignments.

3. RELATED WORK

We classify the related work into three branches.

The BMP Problem: The first branch is the *Bottleneck Matching* problem [14, 4, 9, 11] (BMP). BMP was first proposed by Gross in [14]. Given two sets of n objects, $A = \{a_1, a_2, ..., a_n\}$ and $B = \{b_1, b_2, ..., b_n\}$, and the cost matrix $C_{n \times n}$ (c_{ij} represents the cost of matching a_i with b_j for $1 \le i, j \le n$), BMP is to find the perfect matching between A and B, which minimizes the maximum matching cost.

One may come up with the following straightforward solution to solve our SPM-MM problem by using the existing solutions for BMP. Specifically, we duplicate each o in O $o.w$ times and each p in P $p.w$ times. Then, we can use the existing algorithm originally designed for BMP to find the solution for our SPM-MM problem. However, this duplication is cumbersome and undesirable (especially when the capacities/demands are very large), because the resulting datasets would be prohibitively large.

Next, we describe the most popular solution for BMP. [4] provides a comprehensive study of the solutions of BMP, among which, the *Threshold* method has the lowest time complexity. The best-known algorithm for BMP is due to Gabow and Tarjan in [11], which is based on *Threshold*. *Threshold* is based on the property that the minimized *maximum matching cost* (i.e., the optimal *mmd*) must reside in the cost matrix $C_{n \times n}$. Therefore, it maintains a set X containing the candidates of the optimal *mmd*, which is initialized to be $\emptyset$. For each cost entry c in $C_{n \times n}$, it first constructs a *bipartite graph* between A and B containing the edges each of which is a pair (a_i, b_j) whose matching cost is *at most* c. Then, it checks whether there exists a perfect matching in this bipartite graph. If yes, it includes c in X. Finally, it returns the smallest cost in X, which is shown to be the optimal *mmd*. The above checking operation could be accomplished with a *maximum cardinality matching* procedure [16] on the corresponding bipartite graph which finds the greatest number of matches in the graph. However, *Threshold* incurs an expensive space cost of $O(n^2)$ since it has to maintain the cost matrix $C_{n \times n}$. Thus, it is not scalable to large datasets.

There is an existing study [9] for BMP in the context of spatial databases where the matching distance between two objects is their Euclidean distance. The method in [9] is exactly the *Threshold* algorithm except that the *maximum cardinality matching* procedure [16] is improved. However, this method cannot be used directly for SPM-MM where the capacities/demands are any positive integers. Besides, the techniques in [9] originally designed for improving the maximum cardinality matching procedure in *Threshold* cannot be adopted for our *Threshold-Adapt* algorithm (which will be introduced in Section 4) since *Threshold-Adapt* involves no maximum cardinality matching procedure.

A *monochromatic* version of BMP (i.e., only one set of data) is considered in [5, 10]. But, these studies are different from ours which uses a *bichromatic* setting where two sets of data (i.e., P and O) are considered for matching.

Some recent papers [8, 26] in the field of operations research also studied the bottleneck problem and its variations, but they do not focus on the efficiency issue. Specifically, a common technique in this field [8, 26] is constrained optimization/programming, which is known to be slow for large datasets. Besides, [8, 26] only studied the problems in the context of graphs instead of spatial databases.

Spatial Matching Problems: The second branch is the existing spatial matching problems [25, 22, 21]. [25] proposed the *SPatial Matching* problem (SPM), which generates a *fair* assignment between P and O. [22] proposed the *Capacity Constrained Assignment* problem (CCA), which returns the *globally optimized* assignment. Recently, a continuous version of CCA [21] was proposed where customers move *dynamically*.

Since SPM and CCA have different optimization criteria from SPM-MM, the existing solutions developed for SPM and CCA cannot be applied here. In fact, as will be verified in our empirical study, the mmd's of the assignments of SPM and CCA are much larger than the mmd of the SPM-MM assignment.

Problems with Minimum Maximum Distance: The third branch is related to some other problems [15, 3] using the *minimum maximum distance* as a measurement. Given n cities, the k*-center* problem [15], one of the traditional computer science problems, is to build k warehouses at different cities ($k \le n$) such that the maximum distance from a city to its *nearest* warehouse is minimized. The goal of k-center which is to *select* k cities out of n cities is different from that of SPM-MM which is to *match* service-providers and customers. [3] studied an assignment problem between servers and clients. The matching distance between a server and a client depends on both the physical distance and the *load* of the server where the load of a server corresponds to the number of clients served by this server. In other words, the matching distance between a server and a client defined in [3] in an assignment can be different from the one in another assignment.

4. ALGORITHM THRESHOLD-ADAPT

4.1 Theoretical Properties

Given a set P of service-providers and a set O of customers, let d_o be the optimal *mmd* for the SPM-MM problem. Intrinsically, d_o is a pairwise distance between a service-provider p in P and a customer o in O. It follows that $d_o \in S$, where S is the set of all possible pairwise distances between P and O, i.e., $S = \{d(p, o) | p \in P, o \in O\}$. Note that $|S| = |P| \cdot |O|$. We present this property in the following Lemma 1.

LEMMA 1 (SEARCH SPACE). *Let d_o be the optimal* mmd *for the SPM-MM problem. d_o is in S.* □

According to Lemma 1, one straightforward method of finding d_o is to determine whether each value in S is *feasible* for the SPM-MM problem, insert all feasible values into a set X, and find the minimum value in X as d_o. The definition of "feasibility" is defined next.

DEFINITION 1 (FEASIBILITY). *Given a positive real number d, d is* feasible *if and only if there exists a full assignment A between P and O such that its* mmd *is at most d.* □

LEMMA 2 (FEASIBILITY). *Let d_o be the optimal* mmd *for the SPM-MM problem. d_o is feasible.* □

4.2 Algorithm

We develop our *Threshold-Adapt* algorithm by using the search space S and the feasibility property described in Lemma 2. Specifically, *Threshold-Adapt* checks the feasibility of each distance in S and returns the smallest feasible distance.

THEOREM 1. *The* Threshold-Adapt *algorithm returns the optimal assignment for the SPM-MM problem.* □

Let α be the cost of checking the feasibility of a given value d. One straightforward implementation of *Threshold-Adapt* has the time complexity equal to $O(|S| \cdot \alpha) = O(|P| \cdot |O| \cdot \alpha)$. In the following, we consider two issues of *Threshold-Adapt*.

The first issue is to further reduce the size of the search space from $|P| \cdot |O|$ to $O(\log(\max\{|P|, |O|\}))$ based on the following *monotonicity* property.

LEMMA 3 (MONOTONICITY). *Let d and d' be two positive real numbers where $d < d'$. If d is feasible, then d' is feasible.* □

According to the above lemma, if we know that a value d' in S is not feasible, then any value d in S smaller than d' must not be feasible. This gives hints for a further reduction of the search space.

Specifically, we sort all values in S in ascending order and store the sorted values in a list L. Then, we adopt *binary search* to find the smallest feasible value in L (which corresponds to the optimal *mmd*). This method checks $O(\log |L|)$ pairwise distances in S. Note that $|L| = |P| \cdot |O|$. Thus, the size of the search space becomes $O(\log(|P| \cdot |O|)) = O(\log(\max\{|P|, |O|\}))$ which is significantly smaller than the original size of $|P| \cdot |O|$.

The second issue is to propose an efficient method to perform the judging task to determine whether a given value d is feasible or not. We propose the following three-step algorithm.

Step 1: Construction of a Flow Network wrt d. We construct a *flow network* $G_d(V_d, E_d)$ wrt d as follows. We create a source vertex s and a sink vertex t, and V_d is constructed to be $P \cup O \cup \{s, t\}$. For each pair $(o, p) \in O \times P$ with $d(o, p) \leq d$, we create an edge (p, o) in E_d and set its capacity to be $\min\{p.w, o.w\}$. For each p in P (o in O), we create an edge (s, p) ((o, t)) in E_d and set its capacity to be $p.w$ ($o.w$).

Step 2: Construction of a Maximum-Flowed Network. We perform a *maximum-flow algorithm* [2], denoted by $A_{max-flow}$, on the flow network G_d and obtain the *maximum flow* from s to t in G_d. We denote the amount of this maximum flow by mf. The *maximum-flowed network* is the flow network G_d, where each edge is associated with its flow in the resulting maximum flow. We denote by $e.f$ the flow associated with the edge e.

Step 3: Feasibility Checking on d. We compare mf with W_O. If $mf = W_O$, we conclude that d is feasible; otherwise, we conclude that d is not feasible. In the former case, we construct an assignment, denoted by A_d, based on the maximum-flowed network at Step 2. We initialize A_d to $\emptyset$. Then, for each edge e in the form of (p, o) in the maximum-flowed network with $e.f > 0$, we create a match $(o, p, e.f)$ in A_d.

The correctness of the above three-step algorithm is verified by the following lemma.

LEMMA 4. *The three-step algorithm returns a full assignment A_d with its* mmd *at most d if and only if d is feasible.* □

Time Complexity. After we address the first issue and the second issue, we know that the *Threshold-Adapt* algorithm triggers $O(\log(\max\{|P|, |O|\}))$ times of running the maximum-flow algorithm. Thus, the time complexity of *Threshold-Adapt* is $O(\log(\max\{|P|, |O|\}) \cdot \alpha)$ where α is the cost of a maximum-flow algorithm (e.g., $\alpha = O(n^2 m)$ on a flow network with n vertices and m edges if the recently proposed *IBFS* algorithm [13] is adopted). We will test different maximum-flow algorithms in our experiments for optimizing the performance of *Threshold-Adapt*.

We note here that *Threshold-Adapt* suffers from two intrinsic space problems which limit the application scope of *Threshold-Adapt* to small/medium-sized datasets only. First, it relies on a search space S whose size is $|P| \cdot |O|$. This is prohibitively large when the datasets are large (e.g., S simply occupies about 7.45GB space when $|O| = 100k$ and $|P| = 10k$). Second, it has to maintain a flow network $G_d(V_d, E_d)$ which has its worst-case space complexity of $O(|P| \cdot |O|)$. Motivated by the above space issues of *Threshold-Adapt*, we design another algorithm called *Swap-Chain* in the next section, which not only avoids these issues by adopting a fundamentally different idea, but also runs faster by orders of magnitude.

5. ALGORITHM SWAP-CHAIN

In Section 5.1, we give an overview of the *Swap-Chain* algorithm. We then present it in Section 5.2, and discuss some issues of *Swap-Chain* and its theoretical results in Section 5.3.

5.1 Overview

Swap-Chain has the following three steps.

- **Step 1 (Assignment Initialization):** It first initializes a full assignment A using a given strategy. We will discuss different strategies for this step in Section 5.3. One strategy is finding a fair assignment (which is full) by an existing algorithm [25].
- **Step 2 (Assignment Adjustment):** It re-assigns some matches in A to form another full assignment A' such that the *mmd* of A' is smaller than that of A.
- **Step 3 (Iterative Step):** It repeats Step 2 until it is not possible to perform the assignment adjustment step.

In Step 2, the algorithm reduces the *mmd* of an assignment A by re-assigning some matches in the assignment. Note that the *mmd* of an assignment denotes the maximum matching distance of a match in the assignment and this match is called an *extreme match*. Specifically, the main idea of Step 2 is to find an extreme match in the assignment, break this match and *some* other matches, and re-assign these matches such that the *mmd* of the resulting assignment is smaller.

5.2 Algorithm

5.2.1 Concepts and Algorithm

Before introducing the *Swap-Chain* algorithm, we introduce some concepts and lemmas related to the algorithm.

Let A be an assignment. Given a customer $o \in O$, the *deficient demand* of o in A is defined to be $o.w - \sum_{(o,p,w) \in A} w$. o is said to have his/her deficient demand in A if the deficient demand of o in A is non-zero. Otherwise, o is said to have no deficient demand in A. Given a service-provider $p \in P$, the *free capacity* of p in A is defined to be $p.w - \sum_{(o,p,w) \in A} w$. Similarly, p is said to have its free capacity or have no free capacity in A according to different cases. A service-provider p is said to be *available* in A if it has its free capacity in A. Otherwise, it is said to be *occupied* in A.

DEFINITION 2 (d-AVAILABLE/OCCUPIED SERVICE-PROVIDER). *Given a non-negative real number d and a customer o, a service-provider $p \in P$ is said to be a* d-available service-provider *(*d-occupied service-provider*) for o in A if and only if p is available (occupied) in A and $d(o, p) < d$.* □

EXAMPLE 1. [d-Available/Occupied service-provider] Consider Figure 3(b). For the ease of illustration, we assume that the capacity (demand) of each service-provider (customer) is 1 in the figure. Suppose that we have a (non-full) assignment A equal to $\{(o_1, p_2), (o_3, p_3)\}$. p_1 is an available service-provider in A but both p_2 and p_3 are occupied service-providers in A. Let $d = 10$. Since $d(o_1, p_1) = 5 < d$, p_1 is a d-available service-provider for o_1 in A. Besides, since $d(o_2, p_2) = 7 < d$ and $d(o_2, p_3) = 4 < d$, both p_2 and p_3 are two d-occupied service-providers for o_2 in A. However, since $d(o_2, p_1) = 10$ which is *exactly* equal to d, p_1 is not a d-available service-provider for o_2 in A. Note that there does not exist any d-available service-provider for o_2 in A. □

DEFINITION 3 (d-SATISFIABILITY). *Given a non-negative real number d and a customer o, o is said to be* d-satisfiable *in A if and only if one of the following conditions is satisfied.*

Algorithm 1 Algorithm *Swap-Chain*(P, O)

```
1: initialize a full assignment $A$ between $P$ and $O$
2: while there exists an extreme match $m$ in $A$ which involves a customer
   $o$ such that $o$ is $d$-satisfiable in $A - \{m\}$ where $d$ is the matching
   distance of this extreme match do
3:     $A \leftarrow Swap(A, m)$
4: return $A$
```

- ***Availability Condition:*** *There exists a d-available service-provider for o in A, or*
- ***Non-Availability Condition:*** *There does not exist any d-available service-provider for o in A and there exists a d-occupied service-provider p' for o in A such that p' is matched with another customer o' in A and o' is d-satisfiable in A. In this case, (p', o') is said to be a* d-substitute pair *for o in A.* □

Note that "d-satisfiability" is a recursive definition. The *availability condition* corresponds to the base condition in the recursive definition while the *non-availability condition* corresponds to the recursive condition.

EXAMPLE 2. [d-satisfiability] Consider Example 1. Suppose that the assignment A is still $\{(o_1, p_2), (o_3, p_3)\}$. Let $d = 10$. o_1 is d-satisfiable since there exists a d-available service-provider for o_1 in A (i.e., p_1). o_2 is also d-satisfiable because there does not exist any d-available service-provider for o_2 in A and there exists a d-occupied service-provider for o_2 in A, namely p_2, such that p_2 is matched with another customer o_1 and o_1 is d-satisfiable in A. Thus, (p_2, o_1) is a d-substitute pair for o_2 in A. □

The following lemma shows the relationship between "d-satisfiability" and the optimal assignment for SPM-MM.

LEMMA 5 (OPTIMAL ASSIGNMENT). *Let A be an assignment. If there does not exist any extreme match m in A such that the customer originally matched in m is d-satisfiable in $A - \{m\}$ where d is the matching distance of m in A, then A is the optimal assignment for the SPM-MM problem.* □

The above lemma motivates us to design *Swap-Chain* as shown in Algorithm 1. In this algorithm, *Swap* is the *re-matching operation* related to an extreme match m in A. We will describe how we perform this operation next.

5.2.2 The Swap Operation

We first need to introduce a concept called "d-swapping chain" which is used for the *Swap* operation. Roughly speaking, it is a *list* of objects describing which customers and service-providers in the current assignment are involved in the re-matching (or *Swap*) operation such that the new matching distance for each of these customers is smaller than d where d is a non-negative real number.

A *list* is represented in the form of $(x_1, x_2, ..., x_l)$ where x_i is an object (either a customer or a service-provider) for $i \in [1, l]$ and l is the number of objects in the list. Given a list L in the form of $(x_1, x_2, ..., x_l)$, a pair in the form of (x_i, x_{i+1}) is said to be an *even pair* in L if i is divisible by 2. Otherwise, it is said to be an *odd pair* in L. Given two lists L_1 and L_2 where L_1 is $(x_1, x_2, ..., x_l)$ and L_2 is $(y_1, y_2, ..., y_{l'})$, the *list concatenation* of L_1 and L_2, denoted by $L_1 \diamond L_2$, is defined to be $(x_1, x_2, ..., x_l, y_1, y_2, ..., y_{l'})$.

DEFINITION 4 (d-SWAPPING CHAIN). *Let A be an assignment. Suppose that o is d-satisfiable in A. We define a d-swapping chain from o in A, denoted by $C_d(o)$, as follows according to the availability condition and the non-availability condition.*

- *$C_d(o)$ is the list (o, p') if the availability condition is satisfied where p' is a d-available service-provider for o in A, or*
- *$C_d(o)$ is the list $(o, p') \diamond C_d(o')$ if the non-availability condition is satisfied where (p', o') is a d-substitute pair for o in A.* □

EXAMPLE 3. [d-Swapping Chain] Consider Example 1. The assignment A is still $\{(o_1, p_2), (o_3, p_3)\}$. Let $d = 10$. A d-swapping chain from o_1 in A, denoted by $C_d(o_1)$, can be (o_1, p_1). Besides, a d-swapping chain from o_2 in A, denoted by $C_d(o_2)$, can be $(o_2, p_2) \diamond C_d(o_1)$ (which is equal to (o_2, p_2, o_1, p_1)) since (p_2, o_1) is a d-substitute pair for o_2 in A. □

Let A be an assignment and d be a non-negative real number *at least* the *mmd* of A. Given a customer o, a d-swapping chain from o in A, denoted by C, has the following properties.

- The total number of objects in C is even.
- C is a list containing interleaved customers and service-providers. The first object in C is a customer. We call it as the *first customer* wrt C. The last object in C is a service-provider p' and the second-to-last object in C is a customer o'. We call p' as the *last service-provider* wrt C. Note that p' is a d-available service-provider for o' in A.
- Each *odd* pair in L is in the form of (o', p') and $d(o', p') < d$.
- Each *even* pair in L is in the form of (p', o') and $d(p', o') \leq d$ (note that $d(p', o')$ is at most the *mmd* while d is at least the *mmd* as has been specified). For each even pair (p', o') in L, there exists a positive integer w' such that $(o', p', w') \in A$ and w' is said to be the *weight* of the even pair (p', o').

Note that given a customer o and a non-negative real number d at least the *mmd* of an assignment A, o is d-satisfiable in A if and only if there exists a d-swapping chain from o in A.

After we describe the d-swapping chain, we are ready to describe how to perform the re-matching operation, *Swap*, based on this chain. For the ease of illustration, we first assume that the capacity (demand) of each service-provider (customer) is 1. We call this assumption the *unit assumption*. After we explain the intuition under the unit assumption, we will relax it. Under the unit assumption, the amount of the service given by a service-provider to a customer in a match is exactly equal to 1. Thus, the weight of each possible even pair in a swapping chain C is equal to 1.

Suppose that we are given a full assignment A. We describe our *Swap* algorithm as follows.

Step (a) (Extreme Match Breaking): We find an extreme match m. Let d be the matching distance of m in A and o be the customer matched in m. We then break this extreme match m in A. That is, we remove m from A and form a new assignment A' (i.e., $A' = A - \{m\}$).

Step (b) (Swapping Chain Finding): We then find a d-swapping chain from o in A', denoted by C.

Step (c) (Chain Breaking): Note that each *even* pair (p', o') in C corresponds to a match in A'. For each even pair (p', o') in C, we break the match (p', o') (or formally (o', p')) in A'. Note that the customer o' in each even pair (p', o') in C has no deficient demand before this step but has his/her deficient demand after this step.

Step (d) (Chain Matching): For each *odd* pair (o', p') in C, we form a match (o', p') in A'. At this moment, the customer o' in each odd pair (o', p') in C has no deficient demand.

Let X be the set of customers involved in the swapping chain C. Note that with the above *Swap* algorithm, the *mmd*, say d', of the resulting assignment involving only the customers in X is smaller than the *mmd*, say d, of the original assignment involving only the customers in X. This is because we make sure that for each odd pair (o', p') in C (which forms a match in the resulting assignment), the distance between o' and p' is smaller than d.

If the original assignment contains *exactly one* extreme match, it is easy to see that the *mmd* of the resulting assignment involving *all* customers is smaller than the *mmd* of the original assignment involving *all* customers. However, it is possible that multiple extreme matches exist in an assignment A which have the same matching distance d. The *mmd* of the resulting assignment involving *all* customers decreases only after we break *all* of these extreme matches.

EXAMPLE 4. [Swap] Suppose that the capacity (demand) of each service-provider (customer) is 1. Consider Figure 3(a) which shows a full assignment $\{(o_1, p_2), (o_2, p_1), (o_3, p_3)\}$. We denote this assignment by A. (o_2, p_1) is an extreme match in A. Let $d = d(o_2, p_1) = 10$. The *Swap* operation based on A and match (o_2, p_1) works as follows. First, we break the extreme match (o_2, p_1) and the resulting assignment is shown in Figure 3(b). Second, we find a d-swapping chain C from o_2 which is (o_2, p_2, o_1, p_1) (Refer Example 3 for illustration). Third, we break the even pairs in C which include (p_2, o_1) only. Forth, for each odd pair in C, we form its corresponding match and thus matches (o_2, p_2) and (o_1, p_1) are formed. Figure 3(c) shows the resulting assignment. Clearly, the new assignment is still full, but with a smaller *mmd* (i.e., 7). □

Next, we relax the unit assumption such that the capacity (demand) of each service-provider (customer) could be any positive integer instead of 1. In this case, the weight of an even pair in a swapping chain C can be different from that of another even pair.

The *Swap* algorithm can also be used with this relaxation except the following changes related to the weight of a match.

Step (a) (Extreme Match Breaking): We perform the same operation as before. But, after the breaking of an extreme match in the form of (o, p, w), resulting an assignment A', we obtain that o has its deficient demand equal to w (instead of 1) while p has its free capacity at least w (instead of 1).

Step (b) (Swapping Chain Finding): Similarly, we perform the same operation.

Step (c) (Chain Breaking): In this step, due to the weights of matches, we have to calculate the weights of matches which are used in this chain breaking operation. Specifically, let W be $W_e(C) \cup \{w_o\} \cup \{w_p\}$ where $W_e(C)$ is the set of the weights of all possible even pairs in C, w_o is the deficient demand of the first customer wrt C in A' and w_p is the free capacity of the last service-provider wrt C in A'. We define the *swapping amount* of the chain C, denoted by $Amount(C)$, to be $\min_{w \in W}\{w\}$. Roughly speaking, $Amount(C)$ corresponds to the greatest possible amount of service in a match along the chain such that Steps (c) and (d) can be executed successfully. Let $w_s = Amount(C)$. Note that w_s is smaller than or equal to the weight of each even pair in C.

We execute Step (c) as follows. For each *even* pair (p', o') in C, we break the match (o', p', w') in A' where w' is a positive integer and form a match $(o', p', w' - w_s)$ in A'. Note that the customer o' in each even pair (p', o') in C has no deficient demand before this step but has his/her deficient demand equal to w_s after this step.

Step (d) (Chain Matching): In Step (d), we perform the matching with the weight w_s. That is, for each *odd* pair (o', p') in C, we form a match (o', p', w_s) in A'. Note that at this moment, the customer o' in each odd pair (o', p') in C except the first odd pair has no deficient demand in A'.

Algorithm 2 Algorithm *Swap*(A, m)

```
Input: a full assignment $A$ and an extreme match $m$ in $A$
1: // Step (a) (Extreme Match Breaking)
2: Let $m$ be the match involving customer $o$ and service-provider $p$ with
   its matching distance equal to $d$
3: $A' \leftarrow A - \{m\}$
4: while there exists a $d$-swapping chain from $o$ in $A'$ do
5:    // Step (b) (Swapping Chain Finding)
6:    $C \leftarrow$ a $d$-swapping chain from $o$ in $A'$
7:    // Step (c) (Chain Breaking)
8:    $w_s \leftarrow Amount(C)$
9:    for each even pair $(p', o')$ in $C$ do
10:       find a match $(o', p', w')$ in $A'$
11:       $A' \leftarrow A' - \{(o', p', w')\}$
12:       if $w' \neq w_s$ then $A' \leftarrow A' \cup \{(o', p', w' - w_s)\}$
13:    // Step (d) (Chain Matching)
14:    for each odd pair $(o', p')$ in $C$ do
15:       $A' \leftarrow A' \cup \{(o', p', w_s)\}$
16:    // Step (e) (Deficient Demand Checking)
17:    if $o$ has no deficient demand in $A'$ then break;
18: // Step (f) (Post-Matching)
19: if $o$ has his/her deficient demand in $A'$ equal to $w''$ then
20:    $A' \leftarrow A' \cup \{(o, p, w'')\}$
21: return $A'$
```

Step (e) (Deficient Demand Checking): This step is new. If the customer o in the first odd pair has no deficient demand in A' (this case happens when $w_s = w_o$), we can return the resulting assignment A' generated from Step (d). If o has his/her deficient demand in A' (this case occurs when $w_s < w_o$), then we continue to execute Step (b) to Step (d) until o has no deficient demand in A' or o becomes not d-satisfiable in A'. When we stop the above iterative process, if o has no deficient demand in A', similarly, we can return A' as the output. If o is not d-satisfiable in A', we will run an additional step called *Post-Matching* in Step (f).

Step (f) (Post-Matching): This step is also new. It will be executed if o has his/her deficient demand in A', say w'', and is not d-satisfiable in A'. In this case, it is not possible to reduce the matching distance of the extreme match m involving o. Thus, we create the match (o, p, w'') in A' where p is the service-provider involved in m. Finally, we return A'.

The pseudo-code of *Swap* is shown in Algorithm 2.

With Algorithm 2, it is easy to show the correctness of the *Swap-Chain* algorithm (Algorithm 1) as follows.

THEOREM 2. *The* Swap-Chain *algorithm returns the optimal assignment for the SPM-MM problem.* □

5.3 Remaining Issues & Theoretical Analysis

Remaining Issues. There are two remaining issues in *Swap-Chain*, namely the initialization of a full assignment (line 1 in Algorithm 1) and the Swapping Chain Finding step in the *Swap* algorithm (line 6 in Algorithm 2).

Issue 1: There are many possible ways of initializing a full assignment. In our implementation, we consider the following two methods, namely *Sort* and *Fair*. *Sort* returns an assignment by a two-step approach. First, for each $o \in O$, it maintains a list of all service-providers in ascending order of their distances to o. Second, it processes all $o \in O$ one by one. When processing a specific o, it traverses the service-providers in o's corresponding list sequentially and for the currently traversed p, it assigns the service with the amount equal to $\min\{o.d, p.f\}$ from p to o, where $o.d$ is o's deficient demand and $p.f$ is p's free capacity. The traversing process stops when o's demand has been satisfied. *Fair* denotes the method of generating the fair assignment. Note that we do not adopt the globally optimized assignment in the initialization since the time complexity of the algorithm for finding the globally optimized assignment is much higher than that for finding other assignments like the fair assignment.

Issue 2: For Swapping Chain Finding (i.e., finding a d-swapping chain from a customer o in an assignment A), we design a *Breadth First Search* (BFS) method as follows. It maintains a queue Q and initially inserts o into Q. Then, it processes the elements in Q one by one as follows. It starts processing the first element in Q. If the current element in Q (being processed) is a customer, say o_c, it inserts into Q all service-providers (in any order) that have their distances from o_c *smaller* than d and *have not been inserted* into Q. These service-providers can be found by issuing a range query on P ? from o_c. We say that o_c is the *parent* of all these service-providers. If the current element in Q (being processed) is a service-provider, say p_c, consider two cases. Case 1: p_c has no free capacity. In this case, it inserts all the customers matched with p_c in A into Q (in any order) and p_c is said to be the *parent* of all these customers. Case 2: p_c has its free capacity. In this case, it traces all *ancestors* of p_c until the (starting) customer o is reached, and returns the traced list (in this list, the first element is o and the last element is p_c) as a d-swapping chain from o. The above process continues with the next element in Q until either a d-swapping chain is found or all elements in Q have been processed. In the latter case, it means that there does not exist any d-swapping chain from o.

Here, we need to perform range queries on P. Let $\beta(|P|)$ be the cost of a range query on a dataset of size $|P|$. In [6], with the data structure with its size of $O(|P|(\log|P|\log\log|P|)^2)$ and its construction time complexity of $O(|P|\log|P|)$, $\beta(|P|) = O(\log|P| + k)$ where k is the size of the answer of this query. In practice, $k << |P|$ usually holds. Note that in our implementation, instead of the data structure proposed in [6], we adopt an R-tree index built on P for supporting range queries since it is available in commercial databases and is found to be efficient in practice (though it does not have good worst-case asymptotic performance).

Theoretical Properties. We first describe some theoretical properties which will be used to analyze the time complexity of our *Swap-Chain* algorithm.

Given a match (o, p, w) in an assignment, we say that (o, p) is its *match signature*. Given an assignment A, a list of interleaved objects from P and O in the form of $(o_1, p_1, o_2, p_2 \dots, o_n, p_n)$ is said to be a *match cycle* if each two adjacent objects in the list form a match in A, i.e. o_i is matched with p_i for $1 \le i \le n$, o_{i+1} is matched with p_i for $1 \le i \le n-1$, and o_1 is matched with p_n in the assignment. The length of a cycle is defined to be the number of elements in the cycle. An assignment A is said to be *cyclic* if A contains a match cycle.

Interestingly, a non-cyclic assignment has a theoretical bound on the number of matches in the assignment.

LEMMA 6. *Given P and O, the number of matches in a non-cyclic assignment is bounded by $|P| + |O| - 1$.* □

Furthermore, given an assignment A with a match cycle C, the following lemma suggests that C could be *destroyed* in A easily such that some conditions in A are still satisfied.

LEMMA 7. *Let A be a cyclic assignment with a match cycle C. We can transform A to another assignment A' such that (1) the* mmd *of A' is at most that of A; (2) the deficient demand (free capacity) of each $o \in O$ ($p \in P$) remains unchanged and (3) A' does not contain C nor any matches with* new *match signatures compared with A. Besides, the cost of this transformation is $O(n)$ where n is the length of C.* □

Time Complexity. We let $|V| = |P| + |O|$ and $|E| = |P| \cdot |O|$. Suppose that we build an index as introduced in [6] on P to facilitate range queries described before. Let λ be the time complexity of building this index. Let γ be the time complexity of the full assignment initialization (line 1 of Algorithm 1). Let R be the total number of possible extreme matches fetched in *Swap-Chain* (i.e., the number of iterations in lines 2-3 of Algorithm 1). Let I denote the time complexity of the *Swap* algorithm. The time complexity of *Swap-Chain* is $O(\lambda + \gamma + R \cdot I)$.

Consider λ. From [6], we know that $\lambda = O(|P|\log|P|)$.

Consider γ. If *Sort* is adopted, it could be verified that γ is equal to $O(|O| \cdot |P|\log|P|)$. If *Fair* is used, γ is equal to $O((|P| + |O|) \cdot (\log|P| + \log|O|))$ [25]. Besides, we introduce a lemma which will be used later.

LEMMA 8. *The assignment initialized by* Sort *and the assignment initialized by* Fair *are both non-cyclic.* □

Consider R. Before we give the bound on R, we give a lemma.

LEMMA 9. *A match with a given match signature can be fetched as an extreme match at most once in* Swap-Chain. □

Note that there are at most $|E|$ ($= |P| \cdot |O|$) possible match signatures. By Lemma 9, we deduce that R is bounded by $|E|$. In practice, $R << |E|$. In our experiments, R is about 500 on average, which is very small compared with $|E|$ which is as large as 250,000,000 in our default setting.

Consider I. According to Algorithm 2, I depends on the cost of the while-loop (lines 6-17) and the total number of while-loops, denoted by t. Consider a while-loop which involves the operation of finding a d-swapping chain (line 6), whose cost is denoted by $\mathcal{C}_1$, the operation of re-matching the elements along the chain (line 7-17), whose cost is denoted by $\mathcal{C}_2$, and an additional operation introduced here which is used to transform the assignment obtained to a non-cyclic assignment and whose cost is denoted by $\mathcal{C}_3$. Thus, I is $t \cdot (\mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3)$.

Consider $\mathcal{C}_1$ which corresponds to the time cost of the BFS implementation. Note that at the beginning of each while-loop, the assignment is non-cyclic. This is because the initialized assignment is non-cyclic (Lemma 8) and at the end of each while-loop, the additional operation introduced here transforms the assignment to a non-cyclic one. Thus, it could be verified that $\mathcal{C}_1 = O(|O| \cdot \beta(|P|) + |P|)$ (the BFS method (1) involves at most $|O|$ range queries on P (which incurs the cost of $O(|O| \cdot \beta(|P|))$), and (2) retrieves at most $|P| + |O| - 1$ matches (from service-providers) according to Lemma 6 and the fact that the assignment is non-cyclic (which incurs the cost of $O(|P| + |O|)$)).

Consider $\mathcal{C}_2$. It is simply $O(|P| + |O|)$ (=$O(|V|)$).

Consider $\mathcal{C}_3$. After the Chain Matching step, the assignment contains $O(|V|)$ matches (since it contains at most $|P| + |O| - 1$ matches at the beginning of the while-loop and the Chain Matching step forms at most $\min\{|P|, |O|\}$ new matches). Clearly, each match cycle in an assignment A corresponds to a cycle in the undirected graph $G_A(V', E')$, which involves P and O as vertices in V' and all matches as edges in E'. Note that $|E'| = O(|V|)$ (by Lemma 6). Thus, to find a match cycle in A, we can find a cycle in G_A and this can be easily achieved by a common DFS technique [7], which runs in $O(|V'| + |E'|)$ (=$O(|V|)$) time. According to Lemma 7, destroying a match cycle incurs $O(|V|)$ (a match cycle has its length at most $|P| + |O|$) and it does not introduce any match with a new match signature. So, we can transform the assignment to a non-cyclic one by iteratively destroying the match cycles until no match cycles exist in the assignment. Thus,

$\mathcal{C}_3 = O(c \cdot |V|)$, where c is the number of match cycles formed due to the Chain Matching step. It could be verified that c is bounded by $\min\{|P|, |O|\}$ since the Chain Matching step introduces at most $\min\{|P|, |O|\}$ matches and each such match can form at most one new match cycle. In practice, $c << \min\{|P|, |O|\}$ (e.g., c is about 17 on average in our experiments under the default setting).

Consider t. Recall that t is the number of while-loops in *Swap* needed to re-satisfy the deficient demand of customer o due to the break operation on the extreme match involving o. Clearly, t is bounded by $\overline{w} = \min\{\max_{p \in P} p.w, \max_{o \in O} o.w\}$. Usually, t is much smaller than this upper bound $\overline{w}$. For example, in our experiments on real datasets, on average, t is 2 (with a maximum of 40) but $\overline{w}$ is in thousands.

In view of the above discussion, we know that $I = O(t \cdot (|V| \cdot \beta(|P|) + c \cdot |V|))$. As a result, the time complexity of *Swap-Chain* is $O(\lambda + \gamma + R \cdot t \cdot (|V| \cdot \beta(|P|) + c \cdot |V|))$, where $R << |E|$, $t << \min\{\max_{p \in P} p.w, \max_{o \in O} o.w\}$, and $c << \min\{|P|, |O|\}$.

6. DISCUSSION

Any assignment with its *mmd* equal to the optimal *mmd* is a solution of SPM-MM. Thus, there may exist multiple possible solutions for SPM-MM. In this case, our *Swap-Chain* returns one of them at random. However, SPM-MM can be enriched by considering a secondary objective (e.g., minimizing the sum of the matching distances) for the final solution among these multiple solutions. Furthermore, the bottleneck nature of the SPM-MM objective makes it quite easy to be incorporated with a secondary objective since the optimized mmd, say d_o, can always be used as a *hard* constraint for optimizing the secondary objective. Specifically, matching *any* pair of two objects which has its distance bounded by d_o does not destroy the optimality while matching *any* pair of two objects which has its distance larger than d_o definitely ruins the optimality. Thus, we can adopt a two-step mechanism for the SPM-MM problem with a secondary objective. First, we compute the optimal *mmd*, say d_o, using *Swap-Chain*. Second, we ignore all pairs (o, p) with $d(o, p) > d_o$ for matching when optimizing the secondary objective. For instance, if the secondary objective is to minimize the sum of the matching distances, we can solve this enriched version of SPM-MM easily by first computing the optimal *mmd* d_o and then adopting any popular algorithm for *Minimum Weight Matching* [2] with the constraint that all pairs (o, p) with $d(o, p) > d_o$ cannot be matched (this could be achieved by excluding from the graph used by the algorithm all those edges with the corresponding distances larger than d_o).

Next, we discuss SPM-MM in a more general setting where the pairwise distances between P and O could be *non-metric* or *non-spatial*. Interestingly, our proposed methods can also be adapted to this general setting. *Threshold-Adapt* still works in the general setting (recall that *Threshold-Adapt* is adapted from *Threshold* which is designed for general bipartite graphs). We can also adapt *Swap-Chain* to the general setting with some sacrifice of its time complexity as follows. Two parts involved in *Swap-Chain* rely on the spatial setting, namely the *Fair* method for initializing a full assignment (a fair one) and the BFS method for finding a d-swapping chain in an assignment A. To initialize a fair assignment in the general setting, one can adopt the *Stable Marriage* algorithm which incurs the cost of $O(|P| \cdot |O|)$ [12] (instead of $O((|P| + |O|) \cdot (\log |P| + \log |O|))$ in the spatial setting [25]). To find a d-swapping chain from a customer o in the general setting, one can first *materialize* a *directed* graph $G'(V', E')$ such that (1) $V' = P \cup O$; (2) for each $o \in O$, (o, p) is a directed edge in E' for all $p \in P$ with $d(o, p) < d$; (3) for each $p \in P$, (p, o) is a directed edge in E' for all $o \in O$ that is matched with p in A, and then find a path from o to a service-provider p with its free capacity non-zero using a BFS on G'. The resulting path corresponds to a d-swapping chain from o in A. Clearly, $|V'| = |V|$ and $|E'| \leq |E|$. Thus, the cost of the BFS method is $O(|V| + |E|)$ (instead of $O(|O| \cdot \beta(|P|) + |P|)$ in the spatial setting, where $\beta(|P|)$ is the range query cost on P).

	Cardinality	
	Populated Areas (PA)	Fire Stations (FS)
AB	4,999	447
BC	6,609	595
ON	12,474	1,215
QC	12,936	1,259

Table 1: Real datasets

Factor	Configuration
Cardinality ($\lvert O \rvert$)	10k, 30k, **50k**, 70k, 100k
Dim.	2, **3**, 4, 5
Size ratio (r)	5, **10**, 15, 20, 25
Weight ratio (k)	1, 1.5, **2**, 2.5, 3
O's weights	[1, 10)
Scalability ($\lvert O \rvert$)	250k, 500k, 750k, 1000k

Table 2: Synthetic datasets

Finally, we would like to note some differences between our d-swapping chain technique and the well-known *augmenting path* techniques. A typical augmenting path technique is used for computing the *maximum flow* whose main idea is to iteratively finding an *augmenting path* and augmenting the flow along this path until no augmenting paths are possible. As could be noticed, the goal of an augmenting path technique is to *increase* the flow iteratively while the goal of our d-swapping chain technique is to *keep* the flow while decreasing the *mmd* of the corresponding matching. Specifically, in our d-swapping chain technique, we find an extreme match (o, p) and *break* it so that we will find a d-swapping chain from o, while in an augmenting path technique, there is no such breaking operation on a chosen match before the augmenting path is to be found.

7. EMPIRICAL STUDIES

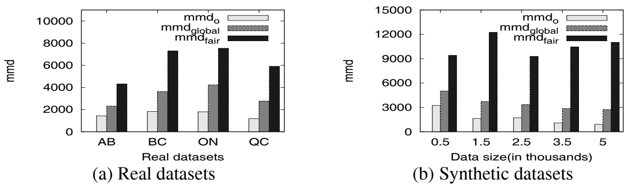
We used four real datasets, namely AB, BC, ON and QC, in our experiments. Each real dataset contains two sets of spatial objects, a set of populated areas (PA) and a set of fire stations (FS). Specifically, dataset AB contains the set of PAs and the set of FSs in Alberta, Canada. Datasets BC, ON and QC contain the same information in the three other provinces in Canada, namely, British Columbia, Ontario and Quebec, respectively. We collected the PAs from Census Canada (http://www12.statcan.gc.ca), each of which corresponds to a dissemination area, and estimated the coordinates of PAs with the help of the Postal Code Conversion File of Canada [20]. The population of each PA ranges from 400 to 700 in most cases [20]. We collected the FSs from FireCanada (http://www.firecanada.ca) and estimated the coordinates of FSs via Google Maps. The capacities of FSs range from 5,500 to 10,000. The coordinates are all normalized to range [0,10000]. For each dataset, we adopt the set of PAs as O and the set of FSs as P. The summaries of the real datasets are shown in Table 1.

We also used synthetic datasets in our experiments, which are generated as follows. The coordinates of spatial objects follow the Uniform distribution on range [0, 10000] by default. The demand of each customer in O is set to be [1, 10) randomly. To generate the capacities of the service-providers in P, we define a parameter k, called *weight ratio*, to be the expected ratio between the sum of the service capacities of all service-providers and the sum of the service demands of all customers, i.e., $k = \sum_{p \in P} p.w / \sum_{o \in O} o.w$. Based on the configuration of k, we set the capacities of the service-providers. By default, the capacities are set to be [80, 120) randomly. The parameter configuration of synthetic datasets is shown in Table 2 where the default values are shown in bold font.

7.1 SPM-MM vs. Existing Spatial Matching

We conducted experiments to compare the optimal mmd, mmd_o, with the mmd's of the fair assignment and the globally optimized assignment, namely mmd_{fair} and mmd_{global}, respectively. In this experiment, we randomly select 10% (5%) in P and 10% (5%) in O for each real (synthetic) dataset. This is because the algorithm (we use the SSPA algorithm in [2]) for computing mmd_{global} is not scalable to large datasets.

Figure 4 shows that mmd_{fair} and mmd_{global} are larger than mmd_o. For example, in the real dataset ON (Figure 4(a)), the ratio between mmd_{fair} (mmd_{global}) and mmd_o is about 3.5 (2.3). We have similar results on synthetic datasets as shown in Figure 4(b).



(a) Real datasets (b) Synthetic datasets

Figure 4: Results for the mmd's of different assignments

7.2 Performance Study

Next, we give the performance study on our proposed algorithms, namely *Threshold-Adapt* and *Swap-Chain*, which include eight instances in total. The details are described as follows.

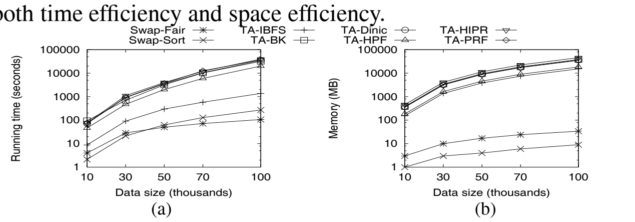
Recall that *Threshold-Adapt* involves a *maximum-flow* procedure. In the literature, many maximum-flow algorithms have been developed which could be categorized into three branches, namely, *Augmenting-Path* (which mainly includes Dinic, BK and IBFS [13]), *Push-Relabel* (which mainly includes HIPR and PRF), and *Pseudoflow* (which mainly includes HPR). More details about these maximum-flow algorithms could be found in [23] (and the references therein). Besides, according to [23], these maximum-flow algorithms usually favor different applications and it is not always the case that a maximum-flow algorithm with a smaller time complexity runs faster than another with a larger one. Motivated by this, we consider all the above six maximum-flow algorithms, namely, Dinic, BK, IBFS, HIPR, PRF and HPF, for optimizing *Threshold-Adapt*, and the corresponding instances of *Threshold-Adapt* are denoted by *TA-Dinic*, *TA-BK*, *TA-IBFS*, *TA-HIPR*, *TA-PRF* and *TA-HPF*, respectively. Besides, we consider two instances of the *Swap-Chain* algorithm, namely, *Swap-Fair* and *Swap-Sort*, with the initialization methods of *Fair* and *Sort*, respectively.

We evaluated the algorithms mainly in terms of *running time* and *memory*, and study the effects of cardinality, dimensionality, size ratio and weight ratio on the performance of the algorithms. The memory of *Threshold-Adapt* is mainly due to the search space S and the flow network graph, and the memory of *Swap-Chain* is mainly due to the R-tree built on P and the maintained assignment.

We implemented our algorithms in C/C++ and conducted the experiments on a Linux platform with a 2.26GHz CPU and 36GB physical memory.

We present our experimental results as follows.

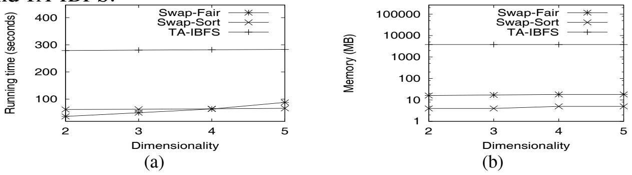
(1) Effect of Cardinality. We vary $|O|$ and the results are shown in Figure 5. We have the following observations. First, there is a clear efficiency gap between the *Swap-Chain* algorithms and the *Threshold-Adapt* algorithms and the gap becomes larger when the data size increases. For example, when $|O| = 100k$, *Swap-Chain* is faster than *TA-IBFS* by more than one order of magnitude. Second, the two *Swap-Chain* algorithms favor different cases. Specifically, *Swap-Sort* runs faster than *Swap-Fair* on relatively small datasets (e.g., $\leq 40k$) while the opposite case becomes true on relatively large datasets. This could be explained by the fact that (1) *Swap-Sort* has no cost of building an R-tree on O while *Swap-Fair* does and (2) *Swap-Sort* has a more expensive initialization procedure (i.e., *Sort*) than *Swap-Fair*. Third, the memory usages of the *Swap-Chain* algorithms are quite low while those of the *Threshold-Adapt* algorithms are dramatically higher (by 2-3 orders of magnitude). For example, when $|O| = 100k$, the *Swap-Chain* algorithms use less than 50MB while each of *Threshold-Adapt* algorithms occupies more than 15GB memory. Forth, among all *Threshold-Adapt* algorithms, *TA-IBFS* runs the fastest and occupies the least memory. For ease of presentation, in the following, we focus on *TA-IBFS* only as the representative of the *Threshold-Adapt* algorithms since it beats all other instances of *Threshold-Adapt* in terms of both time efficiency and space efficiency.



(a) (b)

Figure 5: Effect of cardinality (synthetic datasets)

(2) Effect of Dimensionality. Figure 6 shows the results of the effect of dimensionality. We observe that the dimensionality only affects the *Swap-Fair* algorithm slightly. Specifically, when the dimensionality increases, the running time of *Swap-Fair* increases slightly. This is because *Swap-Fair* needs to build the R-trees on both P (for searching d-swapping chains) and O (for computing a fair assignment), which cost increases when the dimensionality increases. The dimensionality has negligible effects on *Swap-Sort* and *TA-IBFS*.

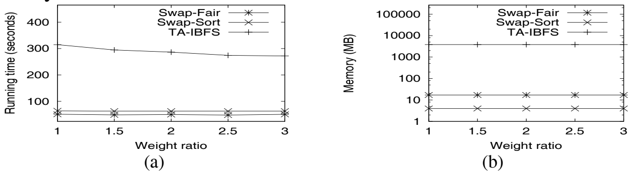


(a) (b)

Figure 6: Effect of dimensionality (synthetic datasets)

(3) Effect of Size ratio. We observe some opposite trends on running time and memory when we increase the size ratio r compared with those when we increase the data size. This is reasonable since when the size ratio r increases, $|P|$ decreases (note that $|O|$ is fixed). Due to limited space, the figures are put in [19].

(4) Effect of Weight ratio. Figure 7 shows the effect of the weight ratio k. We observe that the weight ratio has slight effect on *TA-IBFS* only. Specifically, when k increases, the running time of *TA-IBFS* decreases slightly. The reason might be that when k increases (i.e., the total capacities of the service-providers becomes relatively larger), it is more likely that an augmenting path (note that IBFS is an augmenting path algorithm) *carries* more flow and thus the process of computing the maximum-flow could be finished more quickly.



(a) (b)

Figure 7: Effect of weight ratio k (synthetic datasets)

(5) Scalability test. Figure 8 shows the results of the scalability test for *Swap-Sort* and *Swap-Fair*. Since *Threshold-Adapt* is not scalable, we did not conduct this test for *Threshold-Adapt*. As shown in the figure, the two algorithms are still efficient on large datasets (in millions). Furthermore, *Swap-Fair* is more scalable than *Swap-Sort*. This is because on a large dataset, the initialization process of *Swap-Fair* (i.e., *Fair*) is much faster than that of *Swap-Sort* (i.e., *Sort*).

(6) Experiments on real datasets. Figure 9 shows the results for real datasets which are similar to the results for synthetic datasets.

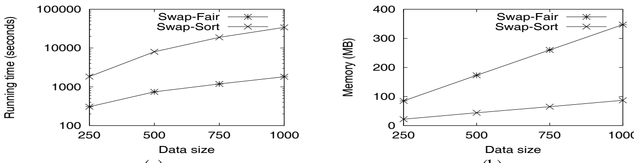


Figure 8: Scalability test (synthetic datasets)

Compared with the *Threshold-Adapt* algorithm, our *Swap-Chain* algorithms run faster and use significantly less memory.

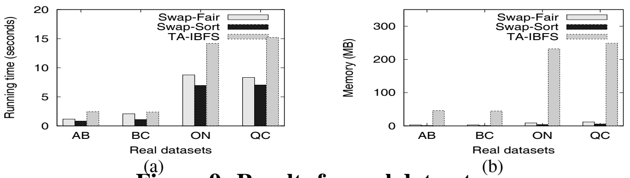


Figure 9: Results for real dataset

(7) Comparison with the *Threashold* algorithm in Euclidean space. We are interested in studying the performance of our proposed algorithms when they are used for the *un-weighted* version of SPM-MM (i.e., all the capacities/demands are 1's). We compare our algorithm with the state-of-the-art called *Match* [9] which has a theoretical time complexity of $O(n^{1.5}\log n)$. We note here that though *Match* has a smaller time complexity, it has quite a narrow application scope (i.e., for the *un-weighted* version only) and the time complexity is restricted to the 2D space only [9]. The results are shown in Figure 10. We observe that our *Swap-Chain* algorithms have comparable running time with the *Match* algorithm and run even faster than *Match* on relatively large datasets. This might be due to the fact that a constant factor which could be large is omitted from the time complexity analysis in [9]. Besides, we found that our *Swap-Chain* algorithms enjoy the superiority of space efficiency over the *Match* algorithm. We also used our real datasets for this experiment by setting the capacities/demands to 1s and observed similar results.

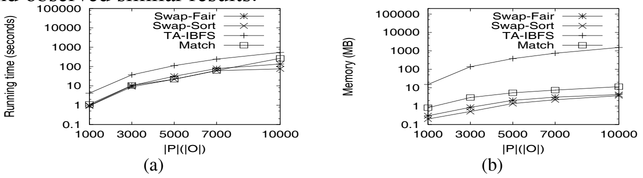


Figure 10: *Threshold* vs. *Swap-Chain*

(8) Experiments with a Secondary Objective. Besides, we conducted some experiments on the SPM-MM problem with a secondary objective of minimizing the *sum of matching distances* called *sum-md*. Let $A_{mmd}(A_{sum-md})$ be the assignment obtained by optimizing *mmd* (*sum-md*) only. Let $A_{mmd,sum-md}$ be the assignment obtained by optimizing *mmd* first and *sum-md* second. We adopted the SSPA algorithm [2] for optimizing *sum-md*. We conducted our experiments on both synthetic and real datasets where each synthetic/real dataset was sampled first with the sampling rate set to 5% due to the relatively expensive cost of SSPA. The results on the real datasets are shown in Figure 11. We observe that on average, compared to A_{sum-md}, $A_{mmd,sum-md}$ can be obtained with a similar time (the cost of optimizing *mmd* is an additional part but the constraint of the optimized *mmd* helps reduce the running time of the process of optimizing *sum-md*) and has the *sum-md* value usually not far away from the *sum-md* value of A_{sum-md} (e.g., within 1.1 factor).

(9) Experiments with non-Euclidean Distances. In addition, we conducted some experiments on our proposed algorithms (*Threshold-Adapt* and *Swap-Chain*) when they are applied to the

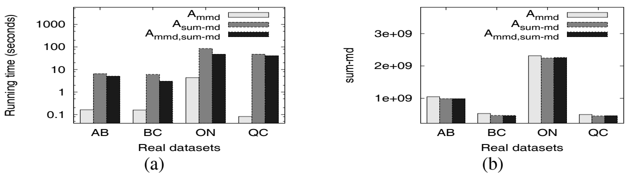


Figure 11: Experiments with a Secondary Objective

cases where non-Euclidean distances are used. We used the same real datasets except that the underlying distances between pairs of two objects are measured by the driving time between the two objects. The results are shown in Figure 12. We observe that compared with the case of using the Euclidean distances, the efficiency of *Threshold-Adapt* is similar while the efficiency of the *Swap-Chain* algorithms, especially *Swap-Fair*, degrades to some extent. But, the *Swap-Chain* algorithms still retain the superiority over *Threshold-Adapt* in terms of both time efficiency and space efficiency. For example, on dataset QC, the running time of *Swap-Fair* (*Swap-Sort*) is about 15s (9s) while that of *Threshold-Adapt* is nearly 17s. Thus, compared with the spatial setting, the degrading ratio of *Swap-Fair* (*Swap-Sort*) is about 9/15 (8/9) and that of *Threshold-Adapt* is about 16/17.

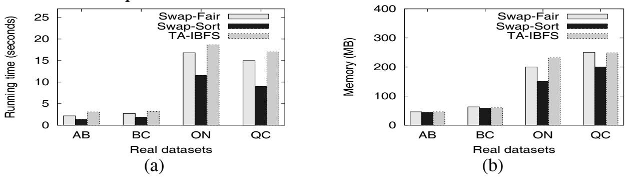


Figure 12: Experiments with Non-Euclidean Distances

(10) Comparison with the Augmenting Path Technique. We also conducted experiments on the *Swap-Chain* algorithms with the adaptions of <u>A</u>ugmenting <u>P</u>ath (AP) techniques [2]. We interpret our d-swapping chains as augmenting paths and denote the resulting *Swap-Chain* algorithms corresponding to *Swap-Fair* and *Swap-Sort* by *AP-Fair* and *AP-Sort*, respectively. In our implementations of *AP-Fair* and *AP-Sort*, when finding an augmenting path which corresponds to finding a d-swapping chain in *Swap-Fair* and *Swap-Sort*, respectively, we do a BFS in a graph structure G which contains the edges (o,p) for all pairs of (o,p) with $d(o,p) < d$ and also the edges (p,o) for all matches (o,p,w) in the current assignment. Figure 13 shows the results. We observe that there is a clear efficiency gap between *Swap-Fair* (*Swap-Sort*) and *AP-Fair* (*AP-Sort*), and this gap becomes larger when the data size increases. Besides, *AP-Fair* (*AP-Sort*) occupies significantly more memory than *Swap-Fair* (*Swap-Sort*). The reason for the efficiency gap is that each range query on P in *Swap-Fair* (*Swap-Sort*) is performed in $O(\log|P|+k)$ time [6] where k is the size of the answer of the query while its counterpart in *AP-Fair* (*AP-Sort*) is performed by scanning an adjacent list which takes $O(|P|)$ time. The reason for the results of memory usage is that *Swap-Fair* (*Swap-Sort*) maintains no graph structures while *AP-Fair* (*AP-Sort*) does.

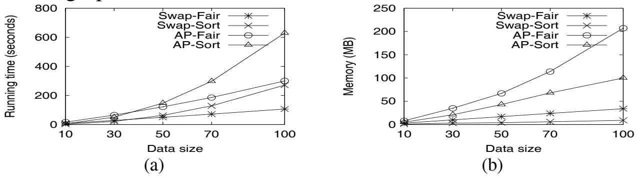


Figure 13: d-swapping chain vs. Augmenting path

Conclusion: The *Swap-Chain* algorithms, which are efficient and scalable, beat the *Threshold-Adapt* algorithms in terms of both time efficiency and space efficiency. Besides, *Swap-Sort* runs faster than *Swap-Fair* when the datset is relatively small (e.g., $|O| \leq 40k$) while *Swap-Fair* enjoys its superiority over *Swap-Sort* on large datasets.

8. CONCLUSION

In this paper, we propose a new problem called *SPatial Matching for Minimizing Maximum matching distance* (SPM-MM). We design two algorithms for SPM-MM, namely *Threshold-Adapt* and *Swap-Chain*. *Threshold-Adapt* is simple and easy to understand but not scalable to large datasets. *Swap-Chain* avoids the scalability issues of *Threshold-Adapt* by adopting a novel idea of swapping the matches iteratively and runs faster than *Threshold-Adapt* by orders of magnitudes. We conducted extensive experiments which verified the efficiency and scalability of *Swap-Chain*. One interesting future direction is to study where to place a new service-provider [24] when we need to minimize the maximum matching distance.

Acknowledgements: We are grateful to the anonymous reviews for their valuable comments on this paper. The research of Cheng Long, Raymond Chi-Wing Wong and Minhao Jiang is supported by grants DAG12EG06S and FSGRF12EG50. The research of Philip S. Yu is supported in part by US NSF through grants IIS-0905215, CNS-1115234, IIS-0914934, DBI-0960443, and OISE-1129076, and US Department of Army through grant W911NF-12-1-0066.

9. REFERENCES

[1] Fire services department, hong kong: Performance pledge 2012. http://www.hkfsd.gov.hk/eng/performance.html.
[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.
[3] E. Bortnikov, S. Khuller, J. Li, Y. Mansour, and J. S. Naor. The load-distance balancing problem. *Networks*, 2010.
[4] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment problems*. Society for Industrial Mathematics, 2009.
[5] M. S. Chang, C. Y. Tang, and R. C. T. Lee. Solving the euclidean bottleneck matching problem by k-relative neighborhood graphs. *Algorithmica*, 8(1):177–194, 1992.
[6] B. Chazelle. New upper bounds for neighbor searching. In *Information and Control*, 1986.
[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. 2009.
[8] T. Dokka, A. Kouvela, and F. C. R. Spieksma. Approximating the multi-level bottleneck assignment problem. In *Operations Research Letter*, 2012.
[9] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.
[10] A. Efrat and M. J. Katz. Computing euclidean bottleneck matchings in higher dimensions. *Information processing letters*, 2000.
[11] H. N. Gabow and R. E. Tarjan. Algorithms for two bottleneck optimization problems. *Journal of Algorithms*, 9(3):411–417, 1988.
[12] D. Gale and L. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69:9–15, 1962.
[13] A. V. Goldberg, S. Hed, H. Kaplan, R. E. Tarjan, and R. F. F. Werneck. Maximum flows by incremental breadth-first search. In *ESA*, 2011.
[14] O. Gross. The bottleneck assignment problem. *The Rand Corporation*, 1959.
[15] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 1985.
[16] J. E. Hopcroft and R. M. Karp. A n5/2 algorithm for maximum matchings in bipartite graphs. In *Switching and Automata Theory*, 1971.
[17] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *ICDE*, 2007.
[18] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *SIGMOD*, 2000.
[19] C. Long, R. C.-W. Wong, P. S. Yu, and M. Jiang. On optimal worst-case matching (technical report). In *http://www.cse.ust.hk/~raywong/paper/spm-mm-technical.pdf*, 2013.
[20] Statistics Canada. Postal code conversion file (pccf). In *http://www5.statcan.gc.ca/bsolc/olc-cel/olc-cel?lang=eng&catno=92F0153X*, 2012.
[21] L. H. U, K. Mouratidis, and N. Mamoulis. Continuous spatial assignment of moving users. *VLDB Journal*, 2010.
[22] L. H. U, M. L. Yiu, K. Mouratidis, and N. Mamoulis. Capacity constrained assignment in spatial databases. In *ACM SIGMOD*, 2008.
[23] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. 2012.
[24] R. C.-W. Wong, M. T. Özsu, P. S. Yu, A. W.-C. Fu, and L. Liu. Efficient method for maximizing bichromatic reverse nearest neighbor. *Proc. VLDB Endow.*, 2(1), Aug. 2009.
[25] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. *VLDB*, 2007.
[26] H. Q. Ye and D. D. Yao. Utility-maximizing resource control: Diffusion limit and asymptotic optimality for a two-bottleneck model. In *Operations Research*, 2010.

APPENDIX

Proof of Lemma 1: d_o is a pairwise distance ($d_o \in S$). □
Proof of Lemma 2: In the SPM-MM assignment which is full, all matching distances are at most d_o. □
Proof of Theorem 1: It follows from Lemma 1 and 2. □
Proof of Lemma 3: Since d is feasible, there exists a full assignment A such that A's *mmd* is at most d and thus A's *mmd* is smaller than d'. Therefore, d' is feasible. □
Proof of Lemma 4: We consider two cases. Case 1: $mf = W_O$. In this case, the three-step algorithm returns A_d. We prove that A_d is a full assignment with its *mmd* at most d. First, we know that A_d satisfies the Capacity Constraint since for each $p \in P$,

$$\sum_{(o,p,w)\in A_d} w = \sum_{e=(p,o)\in E} e.f = (s,p).f \leq p.w$$

Second, we show that A_d satisfies the Demand Constraint by contradiction. Assume there exists a customer $o' \in O$ whose demand is not satisfied in A_d, i.e., $\sum_{e=(p,o')\in E} e.f < o'.w$. We have

$$\begin{aligned} mf &= \sum_{e=(p,o)\in E} e.f = \sum_{o''\in O} \sum_{e=(p,o'')\in E} e.f \\ &< \sum_{o''\in O} o''.w = W_O \end{aligned}$$

which contradicts $mf = W_O$. Third, it is easy to verify that A_d's *mmd* is at most d since for each edge in the form of (p,o) in E, we have $d(p,o) \leq d$ (this is guaranteed by Step 1 of the three-step algorithm). In conclusion, we know that A_d is a full assignment with its *mmd* at most d which further implies that d is feasible. Case 2: $mf < W_O$. In this case, it could be verified that there exists no full assignment which has its *mmd* at most d by contradiction (note that a full assignment implies a flow with its amount equal to W_O which contradicts $mf < W_O$). That is, d is not feasible. □
Proof Sketch of Lemma 5: We consider two cases. The first case is that for each p and each o, $p.w = 1$ and $o.w = 1$. Thus, each match (o', p', w') can be expressed as (o', p') (since $w' = 1$). The second case is a general case that for each p and each o, $p.w$ and $o.w$ can be equal to any positive integer.

Consider the first case. We prove by contradiction. Let A be the assignment such that there does not exist any extreme match m in A such that the customer o originally matched in m is d-satisfiable in $A - \{m\}$ and d is the matching distance of m in A. Suppose that A is not the optimal assignment for the SPM-MM problem. That is, there exists another assignment A_o such that the *mmd* of A_o, denoted by d_o, is smaller than the *mmd* of A, denoted by d. Let $A' = A - \{m\}$.

Consider A_o. We know that for each match $(o', p') \in A_o$, $d(o', p') \leq d_o$. Since $d_o < d$, we have the following. For each match $(o', p') \in A_o$,

$$d(o', p') < d \tag{1}$$

There exists a service-provider p_1 such that $(o, p_1) \in A_o$. We conclude that

$$d(o, p_1) < d \tag{2}$$

Consider A'. We know that o (from match m) is not d-satisfiable in A'. Thus, we deduce that there does not exist any *d-available* service provider for o in A'. We further consider two sub-cases.

Case (a): There does not exist any *d-occupied* service-provider for o in A'. We deduce that there does not exist any service-provider p such that $d(o, p) < d$. This contradicts to that $d(o, p_1) < d$ (in Inequality (2)).

Case (b): There exists a d-occupied service-provider for o in A'. According to Inequality (2), we deduce that p_1 is a d-occupied service-provider for o in A'. Thus, there exists a customer o_1 which is matched with p_1 in A'.

In the following, we will show that o_1 is d-satisfiable in A'. After we obtain this result, we can conclude that o is d-satisfiable in A', which leads to a contradiction.

We first construct an undirected graph G (which will be used later in the proof) as follows. Firstly, we construct an assignment $A_c = (A_o \cup A') - (A_o \cap A')$. As a result, all customers in $O - o$ are involved in either zero matches in A_c or *exactly* two distinct matches in A_c. We construct a set V of vertices to be $P \cup O$. For each $(o', p') \in A_c$, we create an edge (o', p'). All edges created form a set E. The graph G is defined based on V and E.

It is easy to verify that in this graph G, any path starting from o is a list containing interleaved customers and service-providers in the form of $(o, p_1, o_1, p_2, o_2, ...)$ such that the following three rules hold: (R1) o is matched with p_1 in A_o, (R2) o_i is matched with p_{i+1} in A_o for $i = 1, 2, ...$, and (R3) p_i is matched with o_i in A' for $i = 1, 2, ...$

According to the three rules, we deduce the following two statements: (1) any path from o to a service-provider in G is non-cyclic, and (2) there exists a path from o to a service-provider point/vertice p_n such that p_n is the first service-provider with its free capacity in A' along the path. The correctness of Statement (1) can be shown since there is only one edge involving o in E and each vertice in $V - \{o\}$ is involved at most two edges in E. Statement (2) can be proved as follows. Since the total number of vertices is bounded by $|V|$ and any path $\mathcal{P}$ from o to a service-provider is *non-cyclic* (by Statement (1)), the length of $\mathcal{P}$ is *bounded*. Consider a customer o' (not o) along the path $\mathcal{P}$ from o. Since o' is involved in exactly two edges in E (it is not possible that o' is involved in zero edges in E since o' is along $\mathcal{P}$ from o), we know that o' is matched in both A_o and A', and thus the path from o can be prolonged at o'. Consider a service-provider p' along the path $\mathcal{P}$ from o. If p' is involved in exactly two edges in E, similarly, it is matched in both A_o and A', and thus the path from o can be prolonged at p'. If it is involved in exactly one edge in E, it means that it is matched in A_o only (but not A') (by R2) and thus the path from o cannot be prolonged at p'. In this case, p' has its free capacity in A'. This completes the proof when we set $p_n = p'$ in this case.

Based on the above two statements, we conclude that the path is of the non-cyclic form of $(o, p_1, o_1, p_2, o_2, ..., p_{n-1}, o_{n-1}, p_n)$ where p_n is the first service-provider with its free capacity in A' along the path.

Next, we prove that o_i is d-satisfiable in A' for $i = 1, 2, .., n-1$. We prove by induction starting from proving the d-satisfiability of o_{n-1} as a base case. This proof can be done easily by the three rules described above and Inequality (1). For the sake of space, the detailed proof can be found in [19].

Proof of Theorem 2: It follows from Lemma 5. □

Proof Sketch of Lemma 6: First, we show that in a non-cyclic assignment A involving no match cycle, there exists an element e (either a service-provider or a customer) such that e is involved in *exactly* one match in A (We can prove by contradiction since if each element is involved in at least two matches, there exists a match cycle in the assignment). We say this match is *critical*. Second, given an assignment A, we iteratively remove each *critical* match from A until no matches exist in A. Since each removal operation makes at least one element unmatched and the last removal operation makes exactly two elements unmatched, we know the number of matches in A is at most $|P| + |O| - 1$. □

Proof Sketch of Lemma 7: Let the cycle C be $(o_1, p_1, o_2, p_2, ..., o_n, p_n)$. Without loss of generality, let (o_1, p_1) be the match along C which has the smallest matching weight, says w_m. We break each match (o_i, p_i, w_i) with the amount of w_m for each $i \in [1, n]$. Thus, o_i has its deficient demand at least w_m and p_i has its free capacity at least w_m for each $i \in [1, n]$. Next, we create a new match (o_{i+1}, p_i, w_m) for each $i \in [1, n-1]$ and a new match (o_1, p_n, w_m). (Note that for the new match formed (o, p, w_m), if there exists an original match (o, p, w) in the assignment, we just combine these two matches as a single match $(o, p, w_m + w)$). Let A' be the resulting assignment. It can be verified that o_1 and p_1 originally matched in A are not matched in A' and thus A' does not contain cycle C. Besides, it is easy to verify that for each $o \in O$, $\sum_{(o,p,w)\in A} w = \sum_{(o,p,w)\in A'} w$ and for each $p \in P$, $\sum_{(o,p,w)\in A} w = \sum_{(o,p,w)\in A'} w$. Furthermore, the *mmd* of A' is at most that of A and no matches with new match signatures are formed in A'. Clearly, the cost of the above process is simply $O(n)$. □

Proof Sketch of Lemma 8: Let A_s be the assignment initialized by *Sort* and A_f be the one initialized by *Fair*.

We first prove that A_s is non-cyclic. Suppose *Sort* processes O in order of $o_1, o_2, ..., o_m$, where m is the size of O. We denote by A_{o_i} the assignment that is formed immediately after processing o_i (thus, $A_{o_m} = A_s$). We claim that for a specific customer o_i, among all products that are matched with o_i in A_{o_i}, at most one product has its free capacity non-zero. This can be verified by the principle adopted in *Sort* that o_i always exhausts the current product chosen to be matched with o_i before the next product is considered. Now, we show A_s is non-cyclic by contradiction. Assume that there exists in A_s a match cycle $C = (o_{c_1}, p_{c_1}, ..., o_{c_n}, p_{c_n})$. Without loss of generality, among all customers involved in C, we assume o_{c_1} is the first customer processed by *Sort*. Consider $A_{o_{c_1}}$ (the assignment formed immediately after o_{c_1} is processed). Both p_{c_1} and p_{c_n} are matched with o_{c_1} in $A_{o_{c_1}}$ (since p_{c_1} and p_{c_n} are matched with o_{c_1} in A_s) and both of them have their free capacities non-zero (since p_{c_1} (p_{c_n}) is matched with o_{c_2} (o_{c_n}) later on where processing o_{c_2} ($o_c c_n$)). Thus, this leads a contradiction that at most one product matched with o_{c_1} in $A_{o_{c_1}}$ has its free capacity non-zero.

Next, we prove that A_f is non-cyclic by contradiction. This proof can be done by using the fact that no *dangling pair* [25] exists in a fair assignment. The detailed proof can be found in [19]. □

Proof of Lemma 9: This lemma is trivially true if all pairwise distances are distinct. This lemma also holds even if they are not distinct. This is because during the execution of *Swap* (in line 3 of *Swap-Chain*), once the extreme match with a particular match signature is broken, no matches with the same match signature will be formed again except the last Step (f) (i.e., post-processing) which denotes that there is no need to fetch additional extreme matches and the algorithm terminates. □