A SURVEY ON MATCHING PROBLEMS IN SPATIAL DATABASES

By

Cheng Long

A Thesis Submitted to the Graduate

Faculty of Computer Science & Engineering

in Partial Fulfillment of the

Requirements for the Degree of

PhD

Department: Computer Science & Engineering

Approved:

Raymond Chi-Wing Wong, Thesis Adviser

The Hong Kong University of Science & Technology

January 2011

Abstract

Matching is a traditional topic in computer science. Given two sets of objects, we want to match the objects from one set with those from the other set according to some appropriate objectives.

Many matching problems have been proposed in the literature. Some examples are Stable Marriage, Optimal Matching Problem and Generalized Assignment Problem. Furthermore, many algorithms have been designed for these matching problems. Nevertheless, most of them are not efficient enough for large datasets. For instance, the lower bound of the time complexity for Stable Marriage is quadratic and that for Optimal Matching Problem is even worse, cubic. Fortunately, as some researchers discovered, some of these performance limitations do not occur any more when the matching problems are studied in the context of spatial databases.

Recently, considerable research has been conducted on the matching problems in the community of spatial databases. Because the objects involved in many applications of matching are actually spatial entities. Some examples include emergence resource allocation, profile matching and facility location allocation. In this survey, we first review different variants of traditional matching problems and introduce some relevant spatial databases techniques. We then study the matching problems in spatial databases that have been studied recently. Besides, some variants of the matching problems in spatial databases are introduced. Finally, we conclude this survey by giving some future research directions related to matching problems in spatial databases.

Abbreviations

OMP	Optimal Matching Problem
SMP	Stable Marriage Problem
SPM	SPatial Matching Problem
CCA	Capacity Constrained Assignment
OSM	Optimal Stable Marriage
GAP	Generalized Assignment Problem
LOA	Leximin Optimal Assignment
CPR	Closest Pair Retrieval
MCF	Minimum Cost Flow
SSPA	Successive Shortest Path Algorithm
RIA	Range Incremental Algorithm
NIA	Nearest neighbor Incremental Algorithm
IDA	Incremental on-Demand Algorithm
SA	Service-provider Approximation
CA	Customer Approximation
CSA	Continuous Spatial Assignment

Symbols

P, O	Sets of objects
M	A matching between P and O
$c(p, o)$	The cost incurred by pair (p, o)
$c(M)$	The cost incurred by matching M
f	A flow in a graph
$\sigma_p(M)$	The signature of p under matching M
$p.w, o.w$	The capacity of a service-provider p and the demand of a customer o
$dist(p, o)$	The Euclidean distance between p and o
$\alpha(n)(\beta(n))$	The cost of an NN query (object deletion) on a dataset of size n
λ	$\min\{\sum_{p \in P} p.w, \lvert O \rvert\}$
$w(v_1, v_2)(u(v_2, v_2))$	The weight (capacity) of edge (v_1, v_2)
$v.\tau$	The potential of v
$\phi(E)$	The minimum cost of the edges in E
θ	A parameter of RIA
T	The threshold in RIA for a range query
χ	The parameter for bounding the diagonal of MBRs (in CCA)
$score(p, o)$	The score of the pair (p, o) for a preference query

CONTENTS

1. Introduction

Matching is a classical problem in computer science [1, 2, 3]. Let P and O be two sets of objects and M be a subset of *Cartesian product* $P \times O$. Each element (p, o) in M is called a *pair* (*match*). M is said to be a *matching* between P and O if each p (o) occurs in at most one pair of M.

Given P and O, there exist multiple (exponential) matchings between P and O. To obtain a *desirable* matching, we usually impose some *matching objectives* (*matching strategies*) when forming a matching between P and O. Based on different matching objectives, we can form different desirable matchings between P and O. For example, in the *Optimal Matching Problem* (OMP) [1], it is assumed that forming a pair (p, o) would incur a certain cost and the cost of a matching M between P and O is the aggregated cost of all pairs in M. The matching objective of OMP is to *minimize* the cost of the formed matching M and to include in M as many pairs as possible.

Numerous examples of matching exist in real life. For example, job scheduling matches jobs with machines. Facility location allocation involves a matching problem between a set of facilities and a set of locations. Other examples includes matching between students and schools, matching between service-providers (e.g., wireless access points) and customers (e.g., WiFi receivers) and so forth.

We first briefly introduce two basic matching problems in computer science in Section 1.1. Then, we give the motivations for studying the matching problems in spatial databases in Section 1.2. At the end, we provide the roadmap of the following chapters in Section 1.3.

1.1 Matching Problems

In this part, we briefly introduce two classical matching problems in computer science, namely the *Optimal Matching Problem* (OMP) [1] and the *Stable Marriage*

Problem (SMP) [2] in Section 1.1.1 and Section 1.1.2, respectively. The review of other matching problems will be provided in Chapter 2.

1.1.1 Optimal Matching Problem

Let P and O be two sets. For any $p \in P$ and $o \in O$, we use $c(p, o)$ to represent the cost of matching p with o. We say matching M is *optimal* between P and O if M contains as many pairs as possible (i.e., $|M| = \min\{|P|, |O|\}$) and the cost of M is *minimized*. Given P and O, OMP is to find the optimal matching M between P and O.

The matching strategy involved in OMP that optimizes (either maximizes or minimizes) a global measure is very natural and thus it has been used in many real-life applications. For instance, Easychair [1] is a popular software that helps Program Committee (PC) chairs to match referees with papers. Specifically, it asks referees to provide their interests (weights) on the papers at the bidding stage, and then it assigns papers to referees based on these weights such that the *overall weight* (of the referee-paper pairs) is maximized.

1.1.2 Stable Marriage Problem

Let P be a set of men and O be a set of women. Each man in P has a preference list of women in a descending order of how much he likes each woman in O. Similarly, each woman in O has a preference list of men in a descending order of how much she likes each man in P. Assume $|P| = |O|$ (this assumption is made for ease of exposition and can be relaxed easily). We say matching M is *stable* between P and O if $|M| = |P|(|O|)$ (i.e., all men (women) are matched in M) and there exist no *unstable* pairs (p, o). A pair (p, o) is said to be an *usntable pair* if p prefers o to the current woman matched with p in M and o prefers p to the current man matched with o in M. The SMP is to find a stable matching M between P and O.

The underlying matching strategy of SMP is widely used in real life, since it employs a *fair* rule when forming a matching and the requirement of "fairness" is

[1] http://www.easychair.org

very desirable to the participants involved in the matching. Here, the notions of "man" and "woman" are general and can have alternative semantics in different applications. For example, one well-known application of SMP is the school admission system which is to match student applicants with their majors, where student applicants are regarded as men and majors are regarded as women.

1.2 Motivations for Matching in Spatial Databases

The motivations of studying the matching problem in the context of spatial databases are discussed as follows.

First, a lot of algorithms have been developed for different matching problems (e.g., OPM and SMP), though they are not *efficient* or *scalable* enough from the view of databases. To illustrate, consider the aforementioned two popular matching problems, OMP and SMP. For OMP, the best-known solution incurs a *cubic* time complexity (in term of $|P|+|O|$) [4], thus making it prohibitively expensive on *large* datasets of P and O. For SMP, the fastest solution is due to Gale and Shapley [2], which runs in $O(|P|\cdot|O|)$ time and consumes $O(|P|\cdot|O|)$ memory. Clearly, it would be quite time-consuming when dealing with voluminous P and O. Furthermore, it is shown that the complexity bound of $O(|P|\cdot|O|)$ (for both time and space) is indeed the best we can achieve given any arbitrary preference list of each man/woman. However, as will be shown later in this survey, when studied in the context of spatial databases, the matching problems can be solved in a more efficient way due to some spatial properties. More specifically, some of theoretical bounds (e.g., the one for SMP) can be improved and some algorithms (e.g., the ones for OMP) can be speeded up significantly.

Second, we note that many matching problems actually take place in the spatial environment. For example, in the application of emergency facility allocation, we match emergency facilities (e.g., hospitals) with users (e.g., populated estates). In this case, both emergency facilities and users are located in the Euclidean 2D space. Another example is the *profile matching* where matching is formed between

jobs and students. More specifically, each job is represented by several attributes such as the salary, workload and so on. Each student provides his/her preference for these attributes. Thus, each job (student) can be represented by a multidimensional point. In this example, although the elements of P and O seem not to be located in the spatial environment *explicitly*, they can actually be regarded as being located in spatial environment *implicitly* (multidimensional points). For these matching problems where the elements in P and O are spatial objects, the cost of pair (p, o) is usually measured by the Euclidean distance between p and o.

1.3 Organization

The remainder of this survey is as follows. In Chapter 2, we do the literature review on the classical problems in computer science and propose the general framework of matching in spatial databases. In Chapter 3, we survey the related spatial database techniques that have been used in the matching problem in spatial databases. We study two well-studied matchings problems in spatial databases, SPM and CCA, in Chapter 4 and Chapter 5, respectively. Some variants of the matching problem in spatial databases are provided in Chapter 6. In Chapter 7, we conclude our survey by providing some future research directions.

2. Review of Matching Problems

In this chapter, we review some traditional matching problems in Section 2.1. In Section 2.2, we define the framework of matching problems in spatial databases.

2.1 Traditional Matching Problems

We introduce the *Optimal Matching Problem* (OMP) [1] and the *Stable Marriage Problem* (SMP) [2] in Section 2.1.1 and Section 2.1.2, respectively. In Section 2.1.3, we briefly review some other classical matching problem in computer science.

2.1.1 Optimal Matching Problem (OMP)

Let P and O be two sets of objects. We use $c(p, o)$ to represent the cost that occurs when we match p with o. Let M be a matching between P and O. The *cost* of M, denoted by $c(M)$ is defined as the aggregated cost of all pairs in M, i.e., $c(M) = \sum_{(p,o)\in M} c(p, o)$. We say M is an *optimal matching* if it includes as many pairs as possible (i.e., $|M| = \min\{|P|, |O|\}$) and its cost is minimized. We formalize the *Optimal Matching Problem* (OMP) as follows.

Definition 1 (Optimal Matching Problem) *Let P and O be two sets of objects. The* Optimal Matching Problem *is to find a matching M between P and O such that $|M| = \min\{|P|, |O|\}$ and $c(M)$ is minimized.* □

The OMP problem can be solved by reduction to the well-known *minimum cost flow* (MCF) problem [5]. Specifically, we construct a complete bipartite graph G between P and O, and augment it by creating two new vertices s and t, $|P|$ edges (s, p) ($p \in P$) and $|O|$ edges (o, t) ($o \in O$). Then, for each edge (p, o) ($p \in P$ and $o \in O$), we associate with it a cost equal to $c(p, o)$. For each edge involving s or t (i.e., edges (s, p) for all $p \in P$ and edges (o, t) for all $o \in O$), we associate with

it a cost equal to 0. Besides, we associate each edge in the graph with a capacity equal to 1. Let f be the minimum cost flow from s to t on the constructed graph. It is known that the optimal matching M corresponds to the set of such pairs (p, o) where there exists a flow (unit) along edge (p, o) in the constructed graph.

There is rich literature of solutions for the MCF problem. They include adaptations of *primal simplex method* [6], *signature* [7], *relaxation* [8, 9], *Hungarian method* [10, 1] and *Successive Shortest Path Algorithm* (SSPA) [11]. Among these solutions, *Hungarian method* and SSPA are the two with the best worst-case time complexities. Besides, for some special cases of MCF (with integral edge costs), *cost scaling algorithms* are developed [12, 13] to further improve the worst-case time complexities.

2.1.2 Stable Marriage Problem (SMP)

Let P be a set of men and O be a set of women. Each man keeps a preference list of women and each woman has a preference list of men. We define the concept of *stable matching* between P and O as follows.

Definition 2 (Stable Matching) *A matching M is said to be* stable *if there does not exist such a couple (p, o) that p prefers o to the current o' matched with p and o prefers p to the current p' matched with o.* □

The pair (p, o') $((p', o))$ currently in a matching would be *unstable* if p and o prefer each other to o' and p', respectively. This is because p (o) is willing to match o (p), thus breaking (p, o') $((p', o))$. The intuition behind a stable matching M is that it does not allow the occurances of such unstable pairs (p, o).

Given set P of men and set O of women, the *Stable Marriable Problem* is to find a stable matching between P and O.

Definition 3 (Stable Marriage Problem (SMP)) *Let P and O be two sets of objects. Each p (o) has a preference list of o in O (p in P). The* Stable Marriage

Problem *is to find a matching M between P and O such that M is stable and* $|M| = \min\{|P|, |O|\}$. □

The fastest algorithm is due to Gale and Shapley [2]. Specifically, they developed an algorithm for SMP, which runs in $O(|P| \cdot |O|)$ and occupies $O(|P| \cdot |O|)$ space. The algorithm works iteratively as follows.

At the first iteration, each man *invites* his *favorite* women in O. Each woman that has received invitations then chooses her favorite man among all men that have invited her as her *partner candidate*. Note that the partner candidate, say p, of a woman o would be changed if another man p' invites o during the following iterations and o favors p' more than p.

At the second iteration, each man that were rejected at the last iteration now has another chance to invite his *second favorite* woman. Again, each woman that has been invited at this iteration decides her partner candidate by choosing her favorite man among all the men that invite her at this iteration and her current partner candidate (if any).

For the remaining iterations, the algorithm proceeds in the same manner. It stops when no men are rejected or each woman has decided a partner candidate. In [2], the authors prove that the matching consisting of all pairs of a woman and her partner candidate is indeed a stable matching. Since there are at most $|O|$ iterations (the length of a man's preference list), and the cost at each iteration is clearly $O(|P|)$, the overall time complexity of this algorithm is $O(|P| \cdot |O|)$. Besides, it is easy to verify that the space cost of this algorithm is also $O(|P| \cdot |O|)$ (the cost for storing the preference list of each man and woman). Furthermore, it is shown that $O(|P| \cdot |O|)$ is actually the best we can achieve in terms of both time cost and space cost.

Another property worth mentioning here is that given two sets P and O and the preference list of each p in P and each o in O, it is shown in [2] that we can always find a stable matching between P and O. We provide this result in Lemma 1.

Lemma 1 *[2] Given sets P and O, there always exists a stable matching between*

P and O. □

2.1.3 Other Matching Problems

In this part, we review some other matching problems in computer science. They include *Optimal Stable Marriage* [14], *Generalized Assignment Problem* [3], *Rank-Maximal Matching* [15], *Pareto Optimal Matching* [16] and *Leximin Optimal Assignment* [17].

2.1.3.1 Optimal Stable Marriage

The *Optimal Stable Marriage* (OSM) [14] problem is an *enhanced* version the *Stable Marriage Problem* (SMP).

As mentioned in Section 2.1.2, given set P and set O, there exists at least one stable matchings between P and O. Thus, an intuitive question is that "*Can we further define some priorities on the stable matchings?*". In [14], the authors provided such a *priority definition* as follows.

Let P be a set of men and O be a set of women. For each man p, we use $p.rank(o)$ to denote the ranking of woman o on the preference list of p. Here, if p loves o the most, $p.rank(o)$ is defined to be 1. If p loves o the least, it is defined to be $|O|$. Similarly, for each woman o, we use $o.rank(p)$ to represent the ranking of man p on the preference list of o. Let M be a stable matching. If (p, o) is a match in M, we use $partner(p)$ to refer o and use $partner(o)$ to refer p. Then, we define M's *priority*, denoted by $priority(M)$, as $\sum_{p \in P} p.rank(partner(p)) + \sum_{o \in O} o.rank(partner(o))$.

Clearly, for a stable matching M, the smaller $priority(M)$ is, the more desirable M is. Thus, the goal of OSM is to find the stable matching with smallest priority among all stable matchings between P and O. According to [18], the number of stable matchings grows exponentially with $|P| + |O|$, thus, the method of retrieving all stable matchings and comparing their priorities is prohibitively costly. Instead, in [14], the authors proposed a polynomial time ($O(|P| + |O|)^4$) algorithm for OSM.

2.1.3.2 Generalized Assignment Problem

The *Generalized Assignment Problem* (GAP) was first introduced in [3]. Let P be a set of agents and each agent has a budget. Let O be a set of tasks. Each task can be assigned to any agent. Usually, for a task o, different amounts of profits (cost) would be incurred if it is assigned to different agents. The goal of GAP is to assign tasks to agents such that (1) for each agent, the sum of the cost incurred by the tasks assigned to this agent is at most it budget and (2) the amount of overall profits is maximized.

GAP has close relationships with many other popular problems. For instance, when the cost and the profit incurred by each task for an agent are the same as those incurred by the same task for each of the other agents, GAP is similar (identical) to the *Optimal Matching Problem*; when all costs and profits incurred by each task-agent pair, GAP becomes the *Multiple Knapsack Problem*; when there is only one agent, it reduces to the popular *Knapsack Problem*. However, the generic version of GAP is an NP-hard problem [3].

2.1.3.3 Rank-Maximal Matching

The *Rank-Maximal Matching* problem was introduced in [15]. Let M be a matching between P and O. We say M is *rank-maximal* if the maximum number of objects are matched to their *first-choice* objects, and under this condition, the maximum number of objects are matched to their *second-choice* objects, and so forth. It is known [15] that we can find the rank-maximal matching M in $O(\min\{|P| + |O| + C, C \cdot \sqrt{|P| + |O|} \cdot |P| \cdot |O|\})$ time, where C is the largest c such that some objects are assigned to its c^{th}-choice object in M.

2.1.3.4 Pareto Optimal Matching

A matching M is said to be *pareto optimal* [16] if there is no other matching M' such that *some* objects are matched with *better* objects while *no* objects are matched with *worse* objects in M' (compared with M).

We note here that in this problem, it is not necessary that each object keeps

a *complete* preference list of objects in the other set. Specifically, when an object o_1 is not in the preference list of an object o_2 in the other set, it means that o_1 is not acceptable by o_2 and thus we cannot match o_1 with o_2.

The *Maximum Pareto Optimal Matching* problem is to find the pareto optimal matching with the maximum cardinality. It is shown [16] that we can solve this problem with the time complexity of $O(|P| \cdot |O| \sqrt{|P| + |O|})$.

2.1.3.5 Leximin Optimal Assignment

The *Leximin Optimal Assignment* (LOA) problem [17] is defined as follows. Given set P and set O, let M be a matching between P and O (appropriate matching constraints could be imposed on M). For each $p \in P$, we define its *signature* (a real value) under M (based on some appropriate criteria) and we denote it by $\sigma_p(M)$. Let $sort(\sigma_{p_1}(M), ..., \sigma_{p_{|P|}}(M))$ represent the sorted vector of the signatures of $p \in P$ (non-decreasing). Then, the goal of LOA is to find a matching M such that a given appropriate evaluation function on $sort(\sigma_{p_1}(M), ..., \sigma_{p_{|P|}}(M))$ is maximized.

In [17], the authors consider two evaluation functions for $sort(\sigma_{p_1}(M), ..., \sigma_{p_{|P|}}(M))$, namely a *lexicographic function* [17] and a function for the *weighted sum* (of all signatures in the vector). Besides, it is shown [17] that under different settings (problem inputs), the hardness (P or NP-hard) of LOA is different. An instance of LOA about how to assign papers to referees is studied in [17].

2.2 Matching Problems in Spatial Databases

We give an overall study of matching problems in spatial databases in Section 2.2.1. In Section 2.2.2, we propose the general framework of matching problems in spatial databases.

2.2.1 Overview of Matching Problems in Spatial Databases

Recently, considerable concern has been attracted to the matching problems in spatial databases [19, 4, 20]. Given two sets of *spatial* objects, P and O, the

problem is to construct a matching between P and O using some *matching objectives* (*matching strategies*) (similar or identical to those for the traditional matching problems). Usually, in the matching problems in spatial databases, apart from the matching strategies, appropriate *matching constraints* are imposed.

In the sequel, for the matching problems in spatial databases, we relax the definition of *matching* in Chapter 1 as follows. Given two sets P and O, the *Cartesian product* of P and O (i.e., $P \times O$) forms a set of all possible matchings between P and O. Note that in this *Cartesian product*, each object (e.g., p and o) can occur *multiple* times in the pairs of a matching, which is allowed in the matching problems in spatial databases (e.g., SPatial Matching (SPM) [19] and Capacity Constrained Assignment (CCA) [4]). In fact, the constraint about how many times an object (p and o) can occur in the pairs of a matching are regarded as the *matching constraint* of the matching problems.

To illustrate, consider the problem called CCA in [4]. In the CCA problem, each object p in P is called a *service-provider* and each object o in O is called a *customer*. Each service-provider $p \in P$ has a capacity (integer), denoted by $p.w$, indicating the maximum amount of service p can provide. Each customer $o \in O$ is assumed to have his/her demand of service, denoted by $o.w$. In CCA, it is assumed $o.w = 1$ for all $o \in O$. Besides, the cost of pair (p, o) is assumed to be the Euclidean distance between p and o, denoted by $dist(p, o)$. The goal of CCA is to find a matching M between P and O such that (1) no service-provider provides more service than its capacity, i.e., p occurs at most $p.w$ times in pairs of M; (2) no customer receives more service than its demand, i.e., o occurs at most once in the pairs of M; (3) M includes as many matches as possible, i.e., $|M| = \min\{\sum_{p \in P} p.w, |O|\}$; (4) the cost of M is minimized, i.e., $\sum_{(p,o) \in M} dist(p, o)$ is minimized. Conditions (1) and (2) are said to be the *capacity constraint* of the CCA problem.

In other words, CCA is a matching problem in spatial databases which has the goal to minimize the overall cost (matching objective) while satisfying some

requirements (matching constraints).

Consider the problem called SPM in [19] as another example. Assume $|P| \geq |O|$. The goal of SPM is to match each customer o with its *nearest* service-provider in P that has not been exhausted by other *closer* customer. The resulting matching is called a *fair assignment* which will be discussed in Section 4.2.

Again, SPM can be regarded as another matching problem on the objects in spatial databases, which has the goal to find a *fair* assignment (matching objective) while satisfying the condition that each customer must be assigned with a service-provider (matching constraint).

2.2.2 Framework of Matching Problems in Spatial Databases

In view of the above discussion, we propose the general framework of matching problems in spatial databases as follows.

Problem 1 (Framework of Matching Problems in Spatial Databases) *Let P and O be two sets of objects located in the spatial databases (Euclidean space). A matching problem in spatial databases is to find a matching M between P and O such that a specific* matching objective *is achieved and a certain set of constraints is satisfied.* □

The framework above is very general. It can also be applied to each existing matching problem in spatial databases. For example, in an existing problem CCA, the specific matching objective corresponds to the minimization of $c(M)$ while the constraint used corresponds to the capacity constraint.

Within the framework defined in Problem 1, one actually can define arbitrary matching problems in spatial databases by using whatever meaningful matching objectives and/or matching constraints. In the literature of spatial databases, to the best of our knowledge, only two matching objectives have been studied for matching problems. One is to minimize the overall cost (e.g., CCA) and the other is to find a fair assignment (e.g., SPM).

3. Related Spatial Techniques

In this chapter, we study some spatial databases techniques that are related to the matching problems in spatial databases. Specifically, we introduce *Nearest Neighbor Query*, *Reverse Nearest Neighbor Query*, *Mutual Nearest Neighbor Query* and *Closest Pair Retrieval* sequentially in the following sections.

3.1 Nearest Neighbor Query

Nearest Neighbor (NN) query is an old problem in computational geometry [21, 22, 23, 24, 25, 26]. Given a set P of points in 2-dimensional space and a query point q in the same space, the NN query is to retrieve the point in P that has the smallest distance from q. With an index of the points in P that consumes $O(n)$ space, any NN query can be solved in $O(\log n)$ [21] time, where n is the size of P. Several such indexes have been proposed that can fulfill this purpose, e.g., a *trapezoidal map* over the *Voronoi diagram* of P in [21].

Many efforts have also been devoted to performing NN queries on the dynamic datasets. For instance, in [22], a randomized technique is proposed such that each *update* can be handled in $O(\log n)$ time and each NN query can be answered in $O((\log n)^2)$ time where n is the number of points in P when the update/query happens. Recently, Chan [23] provided another tradeoff by showing that one can answer the NN query in $O(\log n)$ time at the cost of $O((\log n)^6)$ for updating.

In the community of spatial databases [27, 24, 25, 26], many heuristic-oriented algorithms were proposed for NN queries. These algorithms, though with high asymptotic complexities, perform quite efficiently in practice due to their effective pruning methods. Another nice feature of these algorithms is that it is easy to extend them to high-dimensional datasets and to apply them to some access methods (e.g., R-tree [27]) which are widely deployed in commercial databases. Among these algorithms, *branch-and-bound* ([24]) and *best-first* ([25]) run the fastest on low-

dimensional datasets while *iDistance* ([26]) is more suitable for high-dimensional datasets.

3.2 Reverse Nearest Neighbor Query

The *Reverse Nearest Neighbor* (RNN) query was first introduced in [28]. Given a set P of points and a query point q, the RNN query is to retrieve all the points in P that take q as their NNs (compared to the other points in the same dataset as q). There are two versions of the RNN query, namely the *Monochromatic RNN* (MRNN) query and the *Bichromatic RNN* (BRNN) query. Specifically, in MRNN, the query point q comes from the same point set P, while in BRNN, the query point q is from another dataset, says Q.

In the following, we focus on BRNN only since MRNN involves a *single* dataset which is significantly different from the matching problems which involve *two* datasets.

The fastest BRNN algorithm is due to [29], which, however, does not support dynamic updates on the datasets. To handle this problem on dynamic datasets, Kang et al. [30] developed a method for maintaining the BRNN results assuming that the updates occur rapidly in form of stream. Xia et al. [31] proposed a problem whose goal is to find the point with the largest BRNN set. [32] studied the RNN query in arbitrary metric spaces. We refer readers to [32] and the references therein for a further detailed study of RNN.

3.3 Mutual Nearest Neighbor Query

There are two types of *Mutual Nearest Neighbor* queries [33, 19]. One is termed with the *monochromatic* mutual NN query and the other is denoted by the *bichromatic* mutual NN query.

The monochromatic mutual NN query was first introduced in [33]. Given a set P of objects, two objects $p \in P$ and $p' \in P$ are said to be *monochromatic mutual NNs* to each other if p is the NN of p' in $P - \{p'\}$ and p' is the NN of p

in $P - \{p\}$. Several solutions have been proposed to find monochromatic mutual NNs [34, 35, 36, 33, 37].

The *bichromatic mutual NN* query is a variant of the monochromatic mutual NN query. Instead of considering a single dataset as the monochromatic mutual NN query, the bichromatic mutual NN query [19] involves two datasets. Specifically, given two sets P and O, two objects $p \in P$ and $o \in O$ are said to be *bichromatic mutual NNs* to each other if p is the NN of o in P and o is the NN of p in O. In [19], the authors proposed an efficient method for retrieving bichromatic mutual NNs.

3.4 Closest Pair Retrieval

Given two sets P and O of objects, *Closest Pair Retrieval* (CPR) is to find the object pair (p, o) whose distance is the smallest among all object pairs in $P \times O$.

Most methods for closest pair retrieval are heuristic-based and do not have good worst-case time complexities [38, 39, 40, 41]. Specifically, none of these methods run in the time with their complexities lower than $O(|P| \cdot |O|)$. We note here that the problem is much easier when P and O are the same dataset and in that case, a method that runs in $O(|O| \cdot \log |O|)$ time has been developed [42].

In [39], the authors consider the problem which *incrementally* retrieves the closest pairs. In particular, the problem reports all possible object-pairs in form of (p, o) in an ascending order of their Euclidean distances. Thus, P and O keep unchanged when the closest pairs are retrieved one by one. This type of closest pair retrieval is called *Inclusive Closest Pair Retrieval* (ICPR).

Recently, *Exclusive Closest Pair Retrieval* (ECPR) was proposed in [43]. Different from the process of ICPR, in ECPR, when a closest pair (p, o) is retrieved, p (o) is removed from P (O) before the next closest pair retrieval is performed. Besides, a dynamic version of ECPR was studied in [43].

4. Spatial Matching Problem

In this chapter, we study one matching problem in spatial databases called *Spatial Matching Problem* (SPM) [19]. Specifically, we explain the motivation of the SPM problem in Section 4.1 and define the SPM problem in Section 4.2. In Section 4.3, we review the solutions of the SPM problem.

4.1 Motivation

As mentioned in Section 3.2, given two sets P and O, the BRNN query on $p \in P$ is to retrieve all $o \in O$ whose nearest neighbor in P is p. BRNN is usually used to decide the "best service". Let P contain a set of service-providers and O correspond to a set of customers. The problem is to, for each p in P, decide the set of customers that p should provide its service to. A natural way is to make p responsible for its BRNN set on O. In this way, each customer o in O would be served by its nearest service-provider.

However, BRNN fails to consider the capacity information of each service-provider. Specifically, some service-providers p might have such a large BRNN set on O that p is not able to serve all the customers in its BRNN set. In this case, we cannot simply assign p to the customers in its BRNN set.

Motivated by this, Wong et al. [19] proposed the *Spatial Matching Problem* which takes into account the capacities of service-providers and assigns to each customer the "best" service-provider it can achieve under the requirement that the resulting assignment is *fair*.

4.2 Problem Definition

Let P be a set of service-providers and O be set of customers in spatial databases. We use $dist(p, o)$ to represent the Euclidean distance between p in P and o in O. Each service-provider p has an integral capacity denoted by $p.w$ and each

customer o has an integral demand denoted by $o.w$. Assume that all service demands of customers can be satisfied by the service-providers, i.e., $\sum_{p \in P} p.w \geq \sum_{o \in O} o.w$.

In [19], two versions of SPM are studied, namely the *un-weighted* SPM and the *weighted* SPM. For ease of exposition, we first review the un-weighted one, where $p.w = 1$ for each $p \in P$ and $o.w = 1$ for each $o \in O$. After that, we will introduce the weighted one, where $p.w$ and $o.w$ can be arbitrary integral numbers. In the sequel, for simplicity, we mean "un-weighted SPM" by "SPM" on default.

Before giving the definition of the un-weighted SPM problem, we first define the concept of *assignment* as follows.

Definition 4 (Assignment) *Let A be a subset of the* Cartesian Product *$P \times O$. A is said to be an assignment if each customer occurs in* exactly *one pair of M (e.g., $|M| = |O|$), and each service-provider occurs in* at most *one pair of M. p and o are said to be* partners *to each other if (p, o) is a pair in M.* □

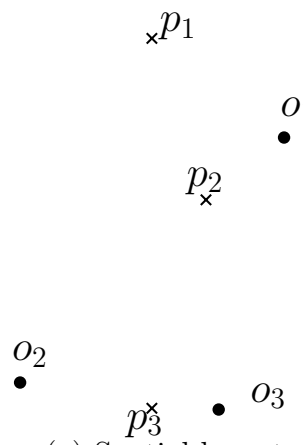
As has been mentioned, the goal of SPM is to find a *fair assignment* given P and O. In [19], the *unfairness* is captured by the concept called *dangling-pair* which is defined as follows.

Definition 5 (Dangling Pair) *Given an assignment A, a couple of objects $(p, o) \in P \times O$ is a* dangling pair *if the following two conditions are satisfied.*

- *$dist(p, o) < dist(p', o)$ where p' is the partner of o;*
- *$dist(p, o) < dist(p, o')$ where o' is the partner of p (this condition is trivially true if p has no partners)*

□

Let A be an assignment and (p, o) be a dangling pair of A. Intuitively, it is *un-fair* to p and o in the sense that both p and o prefer each other to their current partners. Motivated by this, a *fair assignment* is defined to avoid the occurances of



(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

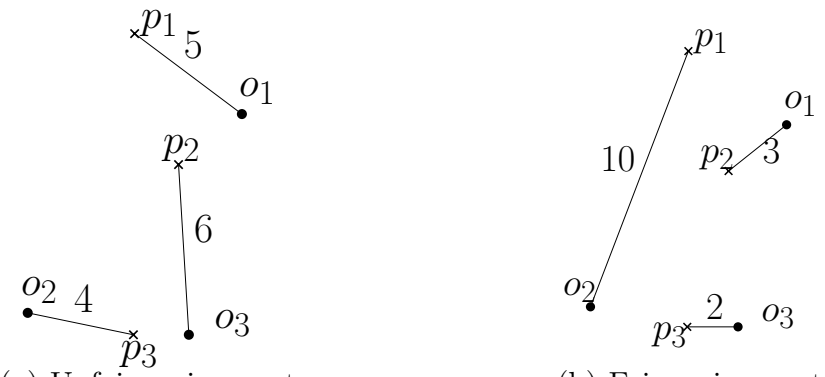
Figure 4.1: A running example for SPM

dangling pairs ("unfairness") in [19]. Specifically, A is said to be *fair* if no dangling pair exists; otherwise, A is *unfair*.

To illustrate, consider the example in Figure 4.1. In Figure 4.1(a), P contains three hospitals p_1, p_2 and p_3 while O includes the same number of residential estates o_1, o_2 and o_3. Figure 4.1(b) shows the information about all pairwise distances between P and O. For the sake of illustration, suppose that the *service-capacity* of each hospital p in P is 1, which means that the greatest amount of the service given by p is 1, while the *service-demand* of each residential estate o in O is 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.

Let us first consider the *assignment* between P and O as shown in Figure 4.2(a). In this figure, 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 corresponds to the pairwise distance between p and o. Consider the pair (p_3, o_3). We know $dist(p_3, o_3) < dist(p_3, o_2)$ and $dist(p_3, o_3) < dist(p_2, o_3)$. As a result, (p_3, o_3) is a dangling pair. Similarly, we know (p_2, o_1) is also a dangling pair. Thus, this assignment is *unfair*. In contrast, the assignment shown in Figure 4.2(b) is *fair*. This is because we cannot find any dangling pairs in this assignment.

The *un-weighted SPM* problem is defined as follows.



(a) Unfair assignment (b) Fair assignment

Figure 4.2: An example for SPM

Problem 2 (Un-weighted SPM) *Given set P and set O,* un-weighted SPM *is to find a fair assignment between P and O.* □

4.3 Solutions

In this part, we introduce the solutions proposed in [19] for the SPM problem. Specifically, SPM is shown to be a special case of the stable marriage problem (SMP) in Section 4.3.1. In Section 4.3.2 and Section 4.3.3, we show that SPM can actually be solved using the techniques of closest pair retrieval and bichromatic mutual NN, respectively. In Section 4.3.4, we introduce the method proposed in [19] which solves SPM with a liner number of NN queries.

4.3.1 Reduction to Stable Marriage Problem

We transform the SPM problem to the SMP in the following way. For each $p \in P$, we create a preference list of all $o \in O$ in the ascending order of their distances from p. That is, p prefers o to o' if $dist(p, o) < dist(p, o')$. Symmetrically, for each $o \in O$, we create a preference list of all $p \in P$ in the ascending order of their distances from o. That is, o prefers p to p' if $dist(p, o) < dist(p', o)$. As a result, a *stable matching* M between P and O that includes $|O|$ pairs corresponds to a *fair assignment* A between P and O [19].

However, as mentioned previously, the best-time complexity of the algorithm

for the SMP is $O(|P| \cdot |O|)$, which is not efficient enough from the point view of databases.

4.3.2 Reduction to Closest Pair Retrieval

As discussed in Section 3.4, given two sets P and O, a *closest pair* query is to find the couple (p, o) such that $dist(p, o)$ is the smallest among all possible couples in the *Cartesian product* $P \times O$. In [19], the authors show that we can find a fair assignment by repeatedly finding the closest pair (p, o) between P and O and removing p (o) from P (O) until O becomes empty (since $|O| \leq |P|$). We form assignment A with all closest pairs we have found. It is shown in [19] that the formed assignment A is *fair*.

Although this method solves the SPM problem correctly, its time cost incurred is expensive. It is easy to note that the above method issues $|O|$ closest pair queries, whereas the closest pair retrieval is a costly process in general. More specifically, the best-known time complexity of finding the closest pair is $O(|P| \cdot |O|)$. As a result, the overall time cost of this method is $O(|P| \cdot |O|^2)$, which is very costly for large datasets.

4.3.3 Reduction to Bichromatic Mutual NN Search

The method of solving SPM via bichromatic mutual NN search relies on the following two lemmas. For simplicity, in the following, we use "mutual NN" to refer "bichromatic mutual NN" on default.

Lemma 2 (Mutual NN) *[19] As long as P and O are not empty, there always exists at least one pair of mutual NNs.* □

Lemma 3 (Reduction to Mutual NN) *[19] Let (p, o) be any pair of mutual pair between P and O. Suppose that we remove p from P and denote the remaining set by P' (i.e., $P' = P - \{p\}$). Similarly, we remove o from O and denote the resulting set by O' (i.e., $O' = O - \{o\}$). Let A' be a fair assignment between P' and O'. Then $A = A' \cup \{(p, o)\}$ is a fair assignment between P and O.* □

Based on the above two lemmas, the authors designed an algorithm for SPM as follows. It iteratively retrieves the mutual NNs (p, o) between P and O and then removes p (o) from P (O) until O becomes empty. All mutual NNs found form an assignment which is shown to be fair in [19].

Clearly, the speed of the above algorithm depends on the efficiency of the mutual NNs search. Existing methods of the mutual NNs search are mainly designed for *monochromatic* scenarios, thus being inapplicable in the above algorithm. An obvious solution for the mutual NNs search is to leverage the method of closest pair retrieval since a closest pair is obviously a pair of mutual NNs. This solution, however, would incur the same expensive time cost as the one introduced in Section 4.3.2 ($O(|P| \cdot |O|^2)$). In fact, as will be shown in Section 4.3.4, finding a pair of mutual NNs is much easier than retrieving a closest pair. Besides, there usually exist multiple pairs of mutual NNs between P and O and searching any one of them is enough in the above algorithm. Motivated by the above discussion, the authors of [19] proposed the *Chain* algorithm which will be introduced in Section 4.3.4.

4.3.4 The Chain Algorithm

The *Chain* algorithm is also based on the method of reducing SPM to performing mutual NNs searches as discussed in Section 4.3.3 but with an efficient method of finding pairs of mutual NNs between P and O. We describe the *Chain* algorithm as follows.

Chain maintains a list C which is used to keep objects in P and O and a set A which is used to store the output (i.e., a fair assignment). Initially, C and A are set to empty. Then, it randomly picks an object $o \in O$ and inserts it into C. After that, *Chain* executes some operations on the list C (which will be discussed in detail next). During the process of the algorithm, whenever C is empty, it checks whether O is empty. If O is empty, it stops; otherwise, it randomly picks an object from O and inserts it into C.

Chain executes operations on the list C by repeatedly checking the *last* object,

denoted by obj, in C. There are two cases for obj.

Case 1: obj is a customer o. In this case, it searches o's NN in P, denoted by p. There are two cases for p. Case 1(a): p is the object just before o in C. In this case, a pair of mutual NNs (p, o) on the current P and O has been found. It then inserts (p, o) into A and removes p and o from the end of C. Besides, it removes p (o) from P (O). Case 1(b): p is not the object just before o in C or there is no object before o in C. In this case, it inserts p into C.

Case 2: obj is a service-provider p. This case is handled in a symmetric way of Case 1. Specifically, it searches p's NN in O, denoted by o. Again, there are two cases for o. Case 2(a): o is the object just before p in C. In this case, a pair of mutual NNs (p, o) on the current P and O has been found. It then inserts (p, o) into A and removes o and p from the end of C. Also, it remove p (o) from P (O). Case 2(b): o is not the object just before p in C. In this case, it inserts o into C.

To illustrate, consider the matching problem shown in Figure 4.1. Assume *Chain* picks o_2 and inserts it into C at the beginning. After that, it searches o_2's (the last object in C) NN in P and finds p_3. Since there is no object before o_2 in C (Case 1(b)), it inserts p_3 into C. Now the last object in C becomes p_3. So, *Chain* continues to search p_3's NN in O and finds o_3. Since o_3 is not the object just before p_3 in C (i.e., o_2), it inserts o_3 into C and the last object in C becomes o_3. Then, it searches o_3's NN in P and finds p_3, which is exactly the object just before o_3 in C. That is, it has found a pair of mutual NN (p_3, o_3), and hence it inserts (p_3, o_3) into A. Finally, it would find the fair assignment as shown in Figure 4.2(b).

The *Chain* algorithm is shown to have several nice features [19]. First, *Chain* mainly performs two types of operations, namely *NN queries* and *object deletions*. Thus, we can leverage the rich literature of NN queries and object deletions to improve the performance of *Chain*. Second, as proven in [19], the number of above two operations are bounded, which is presented in the following Lemma 4.

Lemma 4 (Number of Operations of *Chain*) *[19]* Chain *performs at most* $3|O|$ *NN queries, and exactly* $2|O|$ *object deletions.* □

According to Lemma 4, the number of each type of operations is linear to the size of O. Without loss of generality, let $\alpha(n)$ ($\beta(n)$) represent the cost of an NN query (an object deletion) on a dataset with the size of n. Then, the time complexity of *Chain* is $O(|O| \cdot (\alpha(|P|) + \beta(|P|)))$.

4.4 Extension to Weighted SPM

The *un-weighted* SPM can only handle the situation where $p.w = 1$ for all $p \in P$ and $o.w = 1$ for all $o \in O$. In this part, we study the techniques related to the *weighted SPM* problem which will be defined later.

As a counterpart of *assignment* for the un-weighted SPM, the *weighted assignment* is defined for the weighted SPM as follows.

Definition 6 (Weighted Assignment) *[19] Let A be a set of triplets in form of (p, o, w) where p is a service-provider in P, o is a customer in O and w, an integer, is referred to as the weight of triplet (p, o, w). A is said to be a* weighted assignment *if*

- *for each $o \in O$, the sum of the weights of all triplets in A containing o is exactly equal to $o.w$, i.e., $\sum_{(p,o,w)\in A} w = o.w$;*
- *for each $p \in P$, the sum of the weights of all triplets in A containing p is at most $p.w$, i.e., $\sum_{(p,o,w)\in A} w \leq p.w$.*

□

Let A be a weighted assignment and (p, o, w) be a triplet in A. We say p and o are *partners* to each other.

Again, to capture the "unfairness", we define the concept of *dangling pair* for the weighted SPM as follows.

Definition 7 (Dangling Pair) *[19] Let A be a weighted assignment. We say $(p, o) \in P \times O$ is a* dangling pair *if the following two conditions are satisfied.*

- $dist(p, o) < dist(p', o)$ *where* p' *is one of the partners of* o.
- $dist(p, o) < dist(p, o')$ *where* o' *is one of the partners of* p *(this condition is trivially true if* p *has not been exhausted according to* A*).*

□

Similarly, a weighted assignment A is said to be *fair*, if no dangling pair exists; otherwise, A is said to be *unfair*.

The *weighted SPM* problem is defined as follows.

Problem 3 (Weighted SPM) *Given set* P *and set* O*, the* weighted SPM *is to find a fair weighted assignment between* P *and* O*.* □

Straightforwardly, we can solve the weighted SPM problem by transforming it into an instance of the un-weighted SPM problem as follows. First, we construct two new sets, P' and O'. Specifically, for each $p \in P$, we create $p.w$ copies of p in P' and for each such copy denoted by p', we set $p'.w$ to 1. Similarly, for each $o \in O$, we create $o.w$ copies of o in O' and for each such copy denoted by o', we set $o'.w$ to 1. Thus, we have $|P'| = \sum_{p \in P} p.w$ and $|O'| = \sum_{o \in O} o.w$. Second, we solve the *un-weighted* SPM problem on P' and O', and hence we can obtain a fair assignment denoted by A'. Third, we construct the fair weighted assignment A based on A' by combining all those pairs (p', o') into a triplet (p, o, w) in A if p' (o') is one copy of p (o) and there are w such pairs.

Though, the above adaption of the solutions for the un-weighted SPM can solve the weighted SPM correctly, it incurs an expensive cost due to a large amount of redundant computations. In [19], the authors extended the *Chain* algorithm in a more concise way to handle the weighted SPM. The extended *Chain* algorithm is called *Weighted Chain*.

Similar to *Chain*, *Weighted Chain* proceeds by repeatedly retrieving a pair of mutual NNs. The only difference is presented as follows. In *Chain*, when a pair of mutual NNs (p, o) is found, both of them are removed from the datasets and the list

C (if they exist in C) *definitely* (since p is exhausted and o has been satisfied when we match p with o in the un-weighted SPM). In *Weighted Chain*, however, this is not the case, because it is possible that p is not exhausted or all the demands of o are not satisfied when we match p with o (with a *cetain* amount of the service in the weighted SPM). Specifically, when a pair of mutual NNs (p, o) is found (without loss of generality, we assume o is in C while p is not, since the other case is symmetric), *Weighted Chain* proceeds with two cases.

- Case 1: $o.w > p.w$. In this case, $(p, o, p.w)$ is inserted into A, $o.w$ is decreased by $p.w$, p is removed from P, and p and o are removed from C.
- Case 2: $o.w \leq p.w$. In this case, $(p, o, o.w)$ is inserted into A, $p.w$ is decreased by $o.w$ (if $p.w$ becomes 0, p is removed from P), o is removed from O, and p and o are removed from C.

Similar to *Chain*, it is shown in [19] that *Weighted Chain* involves nice asymptotic features as follows.

Lemma 5 (Number of Operations of *Weighted Chain*) *[19]* Weighted Chain *performs at most* $3(|P| + |O|)$ *NN queries, and at most* $|P| + |O|$ *object deletions.*

□

As a result, it is easy to verify that the time complexity of *Weighted Chain* is $O((|P| + |O|) \cdot (\alpha(|P|) + \beta(|P|) + \alpha(|O|) + \beta(|O|)))$.

5. Capacity Constrained Assignment Problem

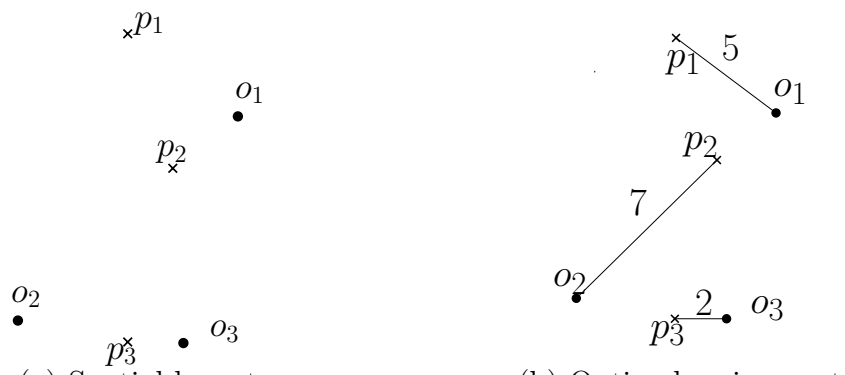
In this chapter, we study another matching problem in spatial databases called *Capacity Constrained Assignment* (CCA) [4]. Specifically, we give the motivation of the CCA problem in Section 5.1 and define the CCA problem in Section 5.2. In Section 5.3 and Section 5.4 , we study exact methods and approximate methods for the CCA problem, respectively.

5.1 Motivation

The CCA problem, which will be defined in Section 5.2, is actually identical to the *Optimal Matching Problem* (OMP) except the following two differences. First, the objects (e.g., p and o) involved in CCA are assumed to be in the spatial context. Second, some objects (objects in P) have arbitrary integral capacities instead of *unit* capacities.

The motivations of CCA are stated as follows. First, there are extensive real-life examples of OMP where the objects involved are indeed located in the spatial context. Thus, it is natural to re-consider the OMP problem with some application domain knowledge about spatial information. Second, as mentioned in Section 2.1.1, the best-known time complexity for the OMP problem is *cubic*, which, however, is prohibitively expensive when *large* datasets are used. Besides, most existing techniques for the OMP problem are main-memory based, and hence they are not suitable for the situations where the data are disk-resided.

In [4], the authors re-considered the OMP (with some adaptions), i.e., CCA, from the perspective of databases. Specifically, they speeded up the traditional techniques for OMP by leveraging some *spatial properties* embodied in the objects involved. Besides, they utilized the techniques in spatial databases such as indexing structure and NN queries to further improve the performance of the proposed methods.



(a) Spatial layout (b) Optimal assignment

Figure 5.1: An Example for CCA

5.2 Problem Definition

Let P be a set of service-providers and O be a set of customers. Each service-provider p has an arbitrary capacity, denoted by $p.w$, while each customer o has a *unit* demand, denoted by $o.w$, i.e., $o.w = 1$ for all $o \in O$. We measure the cost of matching p with o by their Euclidean distance, i.e., $c(p, o) = dist(p, o)$. The cost of matching M is quantified by the overall cost of the pairs in M, i.e., $c(M) = \sum_{(p,o)\in M} c(p, o)$.

The CCA problem is defined as follows.

Problem 4 (CCA) *Given set P and set O, the CCA problem is to find a matching M between P and O such that:*

- *each service-provider p occurs at most $p.w$ times in the pairs of M;*
- *each customer o occurs at most once in the pairs of M;*
- *the cardinality of M is maximized, i.e., $|M| = \min\{\sum_{p\in P} p.w, |O|\}$;*
- *the cost of M, $c(M)$, is minimized.*

In the following, for simplicity, let λ be $\min\{\sum_{p\in P} p.w, |O|\}$. □

To illustrate, consider the example in Figure 5.1. Figure 5.1(a) shows the problem setting. The assignment shown in 5.1(b) is actually the *optimal* assignment

for the CCA problem with the overall cost equal to 14 (5+7+2). That is, we cannot construct an assignment in this problem with a cost smaller than 14.

5.3 Exact Solutions

In this following, we study the exact methods proposed in [4] for the CCA problem. Specifically, we show the method of reducing the CCA problem to the *minimum cost flow* (MCF) problem in Section 5.3.1 and introduce the *Successive Shortest Path Algorithm* (SSPA) for MCF in Section 5.3.2, which acts as a major component of the proposed methods in [4]. In Section 5.3.3, we illustrate the fundamental theorem developed in [4]. We survey all the algorithms for CCA in Section 5.3.4 and discuss some further enhancements for the proposed algorithm in Section 5.3.5.

5.3.1 Reduction to *Minimum Cost Flow*

Similar to the case for OMP, we can solve the CCA problem by reducing it into a *minimum cost flow* (MCF) problem as follows.

First, we construct a graph $G(V, E)$ based on P and O. To construct V, we create a vertex identified by p for each $p \in P$, a vertex identified by o for each $o \in O$ and two additional vertices, source s and destination t. We include all these vertices into V and thus $|V| = |P| + |O| + 2$. To construct E, for each possible couple (p, o), we create edge (p, o) and associate with this edge a cost equal to $c(p, o)$ and a capacity, denoted by $u(p, o)$, equal to 1. Besides, for each $p \in P$, we create an edge (s, p), set its cost $c(s, p)$ to 0 and its capacity $u(s, p)$ to $p.w$. For each $o \in O$, we create an edge (o, t), set its cost $c(o, t)$ to 0 and its capacity $u(o, t)$ to 1.

Second, we calculate the *minimum cost flow* (MCF) from s to t on G, denoted by f. Specifically, the MCF problem on G is to associate each edge (v_1, v_2) with a positive flow denoted by $f(v_1, v_2)$ such that

- $f(v_1, v_2) \leq u(v_1, v_2)$ for each edge $(v_1, v_2) \in E$ (capacity constraint);

- $\sum_{(v_1,v)\in E} f(v_1, v) = \sum_{(v,v_2)\in E} f(v, v_2)$ for each vertex $v \in V - \{s, t\}$ (conservative constraint);
- $\sum_{(s,v)\in E} f(s, v)$ is maximized (and is equal to λ in our case);
- $\sum_{(v_1,v_2)\in E} f(v_1, v_2) \cdot c(v_1, v_2)$ is minimized.

To do this, we can leverage a rich literature of MCF algorithms as discussed in Section 2.1.1.

Third, we construct the matching M for the CCA problem based on f. Specifically, for each edge (p, o) ($p \in P$ and $o \in O$), if there is a flow along this edge, we include the pair (p, o) into M. It can be verified that the resulting matching M constructed above is the solution of the CCA problem [4].

5.3.2 Successive Shortest Path Algorithm

The *Successive Shortest Path Algorithm* (SSPA) is a popular method for the MCF problem [11]. For the graph $G(V, E)$ constructed in Section 5.3.1, the SSPA algorithm proceeds as follows.

For each vertex $v \in V$, it maintains a value called *potential*, denoted by $v.\tau$. For each edge $(v_1, v_2) \in E$, it keeps a *weight*, denoted by $w(v_1, v_2)$. Initially, $v.\tau$ is set to 0 for all $v \in V$, $w(v_1, v_2)$ is set to $c(v_1, v_2)$ for all edge (v_1, v_2) in E and $f(v_1, v_2)$ is set to 0 for all edges (v_1, v_2) in E. It runs with λ iterations.

At each iteration, it computes the *shortest path* from s to t based on the *weight* information of edges using the *Dijkstra* algorithm on the *residual graph* of G wrt the current flow (Refer [5] about how to construct the residual graph wrt a flow). We emphasize here that all notations of *shortest path* or *shortest distance* are based on the weight information of edges. In the sequel, for simplicity, we mean "shortest path from s to t" by "shortest path". Let sp denote the computed shortest path and v_m be the second last vertex along sp. We use $v.\gamma$ to represent the shortest distance from s to v. It then *augments* the flow (unit) along the shortest path sp. After that, for each visited (de-heaped) vertex v during the *Dijkstra* process, it (1)

sets $v.\tau = v.\tau - v.\gamma + v_m.\gamma$; (2) for all edges (v, v_1) that are incident to v, sets $w(v, v_1) = c(v, v_1) - v.\tau + v_1.\tau$.

The resulting flow calculated by SSPA is guaranteed to be the minimum cost flow [11]. We note here that the *costs* of edges keep unchanged during the whole process of SSPA and it is the *weights* of edges that the algorithm keeps updating. The intuition of maintaining the potential (weight) for each vertex (edge) in the SSPA algorithm is that it wants to guarantee that the weights of edges are always *non-negative* which is required by the *Dijkstra* algorithm.

5.3.3 Fundamental Theorem

Since the constructed graph $G(V, E)$ above is nearly a *complete* bipartite graph, applying SSPA directly to the MCF problem on G would incur excessive memory consumption and costly shortest path calculations. To alleviate these space and running time problems, the authors of [4] proposed to use only a subset of E, denoted by E_{sub}, and it can also be guaranteed that at each iteration of SSPA, the computed shortest path based on E_{sub} is identical to the one based on E if a condition based on E_{sub} is satisfied.

Before introducing this technique, we first define the concept of a *distance bounded* edge set as follows.

Definition 8 (Distance-bounded Edge Set) *An edge set $E_{sub} \subseteq E$ is said to be* distance-bounded *if*

$$\forall (v_1, v_2) \in E_{sub}, c(v_1, v_2) \leq \phi(E - E_{sub})$$

where $\phi(E - E_{sub})$ represents the minimum *cost among all costs of edges in $E - E_{sub}$.*

□

Intuitively, a distance-bounded edge set E_{sub} contains only those edges (v_1, v_2) whose costs are below (inclusively) a threshold (e.g., $\phi(E - E_{sub})$). E_{sub} is called

"distance-bounded" (instead of "cost-bounded") is due to the fact that $c(p, o) = dist(p, o)$ for all edges $(p, o) \in E$.

The fundamental theorem based on the concept of a distance-bounded edge set is given in Theorem 1.

Theorem 1 (Fundamental Theorem) *[4] Given a distance-bounded edge set $E_{sub} \subseteq E$, let sp be the shortest path from s to t in E_{sub} and τ_{max} be the maximum potential value (i.e., $\max\{v.\tau | v \in V\}$). If the total cost of sp, denoted by $c(sp)$, is at most $\phi(E - E_{sub}) - \tau_{max}$, then sp is also the shortest path in E (complete graph).* □

The implicit idea behind Theorem 1 is that if the cost incurred by the shortest path in a subgraph E_{sub} is not larger than the smallest *weight* among those weights of all edges in $E - E_{sub}$, then the shortest path in the complete graph E *must* not include any edges in $E - E_{sub}$ (by contradiction).

5.3.4 Algorithms for CCA

According to Theorem 1, at each iteration of SSPA, instead of storing the *complete* graph G with the edge set E and computing the shortest path in it, we can actually maintain a *partial* graph G_{sub} of G with a smaller edge set E_{sub} and compute the shortest path in G_{sub}, which is identical to the shortest path in the complete graph G. In this way, we save both the space for storing the graph and the time for shortest path calculation.

In this part, we study different methods in [4] that utilize the above idea to improve the performance of SSPA. They include *Range Incremental Algorithm* (RIA), *Nearest Neighbor Incremental Algorithm* (NIA) and *Incremental on-Demand Algorithm* (IDA).

5.3.4.1 Range Incremental Algorithm (RIA)

The *Range Incremental Algorithm* (RIA) performs in an identical way as SSPA except that in each iteration, the shortest path *sp* is computed on the subgraph G_{sub} in RIA while on the complete graph G in SSPA.

RIA maintains a threshold T which is used as a lower bound of $\phi(E - E_{sub})$ during the process of RIA and is set to θ initially, where θ is a user parameter in RIA and is a non-negative real number. τ_{max} is used to represent the maximum potential among all potentials of vertices in RIA. Before performing the iterations, RIA initializes E_{sub} by including all edges with costs at most T in E_{sub}. Thus, $\phi(E - E_{sub})$ is at least θ after this initialization.

To ensure that the shortest graph sp computed on the subgraph E_{sub} is indeed the one on the complete graph E, after each shortest path computation, RIA checks whether the cost of sp, $c(sp)$, is at most $T - \tau_{max}$. We call this checking *SP-checking*. There are two cases. Case 1: $c(sp) \leq T - \tau_{max}$. In this case, sp is concluded to be the shortest path on the complete graph according to Theorem 1. Correspondingly, we say this SP-checking *succeeds*. Case 2: $c(sp) > T - \tau_{max}$. We say the SP-checking *fails*. In this case, it keeps performing the following procedures until the SP-checking succeeds. First, RIA updates T with $T + \theta$. Second, it enlarges E_{sub} by including in E_{sub} all edges whose costs are within range $(T - \theta, T]$. Third, it re-computes the shortest path and performs the SP-checking on it again.

It is easy to verify that RIA achieves the same goal as SSPA. The only difference is that RIA *incrementally* enlarges the graph for shortest path computation in the algorithm, while SSPA uses the *complete* graph all the time throughout the algorithm. Thus, RIA saves some space and also running time compared to SSPA.

5.3.4.2 Nearest Neighbor Incremental Algorithm (NIA)

RIA incrementally enlarges the subgraph G_{sub} based on the parameter θ, which, however, is not user-friendly. Specifically, for the case that θ is set too large, the benefits from incrementally maintaining E_{sub} (in G_{sub}) are limited, while for the case that θ is set too small, it has to perform the shortest path computations many times in order to enlarge E_{sub} (in G_{sub}) sufficiently.

Motivated by this, *Nearest Neighbor Incremental Algorithm* (NIA) is proposed to enlarge the subgraph G_{sub} based on *nearest neighbor queries* instead of *range*

queries (RIA). NIA is identical to RIA except that whenever the SP-checking fails, it enlarges E_{sub} by including in it the edge with the minimum cost among all edges in $E - E_{sub}$, computes the shortest path based on the enlarged E_{sub} and do the SP-checking again.

It is shown in [4] that NIA outperforms RIA in terms of running time empirically.

5.3.4.3 Incremental on-Demand Algorithm

In RIA/NIA, when a shortest path sp on the subgraph E_{sub} is computed, the SP-checking is to check whether $c(sp) \leq \phi(E - E_{sub}) - \tau_{max}$, where $\phi(E - E_{sub})$ represents the minimum cost of the edges in $E - E_{sub}$. The implicit idea is that $\phi(E - E_{sub}) - \tau_{max}$ is actually used as the *smallest possible* cost of the path that contains an edge in $\phi(E - E_{sub})$. Clearly, if a more *tighter* bound is used, it is more likely that the SP-checking would succeed.

Motivated by this, the authors in [4] proposed to use another tighter lower bound for the SP-checking. More details can be referred in [4].

5.3.5 Further Enhancements

The first enhancement is re-using the information among consecutive *Dijkstra* processes during the same iteration of NIA/IDA. This technique is not applied to RIA since there might be a bulk of edges inserted into E_{sub} at one time when E_{sub} enlarged, which makes it very difficult to re-use the information generated by *Dijkstra* on E_{sub} before the enlargement for the consecutive *Dijkstra* process on E_{sub} after the enlargement. In NIA/IDA, E_{sub} is enlarged with one edge by one edge, which makes it possible to re-use the *Dijkstra* process information (e.g., heap). Since this technique is more about implementation than theoretical results, we do not provide much details in this survey. Refer [4] for more details.

As can be noted, numerous NN queries are invoked during the processes of the methods introduced above, which incurs excessive cost. To relieve this problem, the authors in [4] proposed the second enhancement, which is to employ an incremental

all-nearest-neighbors techniques. The main idea is that, instead of performing NN queries *separately* and *from scratch* each time, it computes the NN results for all service-providers $p \in P$ *collectively* and *incrementally*.

5.4 Approximate Solutions

In [4], the authors also developed an approximation mechanism for the CCA problem, which is tunable between the *matching accuracy* and the *running time*. Specifically, this approximation mechanism has three phases. The first phase is called *partitioning phase*. In this phase, smaller datasets denoted by P' (O') are created based the original datasets P (O). The second phase is termed as *concise matching*. As implied by the name, it solves the CCA problem by using the smaller datasets P' and/or O' and obtains a matching denoted by M' which is optimal based on P' and/or O'. The last phase is *Refining phase*. In this phase, it constructs a matching M based on the original datasets P and O according to the matching M'. These three phases are introduced in a more detailed way in the following three sections, respectively.

5.4.1 Partitioning Phase

In [4], the objects in P and those in O are assumed to be maintained in different ways. Specifically, set P, whose size is assumed to be much smaller than O's, is stored in main memory, while set O is indexed by an R-tree on disk. As a result, different methods are provided in [4] for partitioning the objects in P and those in O. The method of partitioning the objects in P is called *Service-provider Approximation* (SA) and that of partitioning the objects in O is called *Customer Approximation* (CA). The goal is to partition the objects into different groups such that the diagonal of the *minimum bounding rectangle* (MBR) of each group is at most a user parameter χ. Intuitively, the larger χ is, the smaller number of groups it will obtain.

SA groups the objects in P by scanning the objects in the ascending order of

their *Hilbert values* [44]. For each scanned object p, if there exists a group that can include p while yielding an MBR with the diagonal at most χ, p is included in that group; otherwise, it creates another new group and includes p in this new group.

CA groups the objects in O by traversing the R-tree once. Specifically, it begins at the root. If the underlying node, denoted by N, is an inner-node, we create groups according to the following two cases. Case 1: N's MBR has its diagonal at most χ. In this case, CA simply creates a group and includes in this group all objects in the descendants of N. After that, CA will not traverse the nodes on the subtree of N. Case 2, N's MBR has its diagonal more than χ. CA continues to traverse each child node of N. In the other case where N is a leaf-node. It recursively splits the MBR of N into halves until the diagonal of the half becomes at most χ. Then, for each resulting half, it creates one group and includes in this group all the objects located in the half.

Besides, for each group constructed by SA/CA, a representative p' (o') at the geometric centroid (weighted geometric centroid for SA) of the group is created to represent all the objects in the group. The capacity (demand) of p' (o') is set to the sum of the capacities (demands) of the objects in the group. The new dataset P' (O') corresponds to the set containing all these representatives.

5.4.2 Concise Matching

In the phase, it distinguishes the approximation mechanism into two approximate algorithms. The first algorithm applies the exact methods for CCA on P' and O, and we denote this algorithm by *SA* (P' is used instead of P). The second algorithm applies the exact methods for CCA on P and O', and we denote this algorithm by *CA* (O' is used instead of O). We denote the resulting matching by M' for either SA or CA.

5.4.3 Refining Phase

In this phase, the approximation mechanism constructs the matching on the original datasets P and O, denoted by M, according to M'. The underlying idea is

as follows. If two representatives $p' \in P'$ and $o' \in O'$ are matched with each other, then we can match any object p in the group represented by p' with any object o in the group represented by o' (with the consideration of capacities/demands). In particular, in [4], the authors proposed two heuristics about how to match the objects in the group represented by p' with those in the group represented by o' when p' and o' are matched with each other in M'. They are the *NN-based refinement* and the *exclusive NN refinement*. Refer to [4] for more details.

5.4.4 Approximation Guarantees

Before providing the approximation guarantees of SA and CA, we first define the *approximation error* clearly. Given set P and set O, let M_{opt} and M_{appr} represent the *optimal* solution and the *approximate* solution of the CCA problem on P and O, respectively. Then, the approximation error of M_{appr} is defined to be $c(M_{appr}) - c(M_{opt})$.

It is shown in [4] that SA and CA have approximation guarantees as shown in Theorem 2 and Theorem 3, respectively.

Theorem 2 (Approximation Guarantee of SA) *[4] The approximation error of SA is bounded by* $2 \cdot \lambda \cdot \chi$. □

Theorem 3 (Approximation Guarantee of CA) *[4] The approximation error of CA is bounded by* $\lambda \cdot \chi$. □

Note that the difference between the approximation error of SA and that of CA is due to their different strategies for locating the representatives. Specifically, for each group, SA defines its representative at the *weighted* (using the capacity information) centroid of the rectangle, while CA simply defines the representative at the *geometrical* centroid of the rectangle since each customer is assumed to have demand of 1. More details can be found in [4].

6. Variants of Matching in Spatial Databases

In this chapter, we study three variants of matching problems in spatial databases. They include *Continuous Spatial Assignment of Moving Users* (Section 6.1), *Matching in Preference Databases* (Section 6.2) and *Facility Location Allocation Problem* (Section 6.3).

6.1 Continuous Spatial Assignment of Moving Users

In [20], the authors study the *Continuous Spatial Assignment* (CSA) problem. The CSA problem is identical to the CCA problem except the following two differences.

First, apart from the *capacity constraint*, CSA imposes one more constraint called *coverage constraint* on each service-provider $p \in P$. More specifically, each service-provider $p \in P$ has a *coverage region* and it cannot provide its service to the customers outside its coverage region. Second, CSA considers the scenario where the service-providers are fixed while the customers are moving dynamically. The goal of the CSA problem is to *continuously* report the *optimal* assignment (i.e., similar to the one for CCA but with the coverage constraint) in the dynamic environment.

To handle the CSA problem, the authors of [20] proposed to first initialize an optimal assignment of CSA and then maintain this assignment according to the dynamic changes in the environment.

Specifically, the initialization of the optimal assignment has three steps. The first step makes some *approximate* matches that *must* be included in the optimal assignment by using some geometrical properties. Besides, it excludes some pairs of service-providers and customers that *must not* be in the optimal assignment. The second step then distinguishes different types of service-providers and subsequently decomposes the problem into several smaller ones based on this information. The third step solves each smaller problem instances with appropriate methods.

To model the dynamic environment, three types of events are considered in [20], namely *location update*, *connect request* and *disconnect request*. The authors classify these events into two categories, namely *insertion* and *deletion*. More specifically, an insertion indicates a connect request from a customer, while a deletion means a disconnect request from a customer. A location update can be regarded as a deletion (at the old location) plus an insertion (at the new location). For each event category, a corresponding algorithm for updating the optimal assignment is provided in [20]. Refer the paper for more details.

6.2 Matching in Preference Databases

[45] studies the matching problem between objects and preference queries on the objects. Let P be a set of objects and each object $p \in P$ is represented by a set of attributes that describe the object. Let O be a set of queries and each query is issued with a set of weights (sum to 1) indicating the relative importance of the corresponding attributes. We define the score of query-object pair (p, o), denoted by $score(p, o)$, as the weighted sum of all the attributes of p by using the weights of o. The goal is to find a *fair assignment* M between P and O, that is, there does not exist a query-object pair (p, o) such that

- $score(p, o) > score(p, o')$ where o' is currently matched with p in M (this condition is trivially true if p is not matched);
- $score(p, o) > score(p', o)$ where p' is currently matched with o in M (this condition is trivially true if o is not matched).

Same as the SPM problem, the above matching problem is a *special* instance of the *Stable Marriage Problem* (SMP). We say a query-object pair (p, o) is *stable* if (1) there is no $o' \in O$ such that $score(p, o') > score(p, o)$ and (2) there is no $p' \in P$ such that $score(p', o) > score(p, o)$. It is shown in [45] that this problem can be solved by repeatedly finding *stable* query-object pairs (p, o) and removing them from the problem.

One straightforward method for retrieving a *stable* query-object pair is to do *brute-force* searching. Specifically, it calculates the *score* for each possible pair and pick the pair with the largest *score*, say (p, o). It is easy to verify that the picked query-object pair (p, o) is indeed stable.

Clearly, the above method incurs expensive cost since it has to consider *all* objects p in order to traverse all possible query-object pairs. Instead, the authors of [45] observe that it is sufficient to *only* consider the skyline [46] of P for this purpose. This is because the measure of the *score* is *monotone* to the attributes of objects, i.e., if object p dominates object p', then for any query o, we have $score(p, o) > score(p', o)$.

Motivated by this, the authors in [45] proposed a method which only considers the skyline of P at each stage for stable query-object pair retrieval. Besides, an efficient technique for dynamically maintaining the skyline results is developed. This is because each time the algorithm finds a stable query-object pair (p, o), p is removed from P and thus the skyline of P should be updated accordingly. Furthermore, this skyline maintenance technique is proved to be I/O optimal. More details about this technique can be found in [45].

In [45], the authors considers two variants of this matching problem. The first is more general than the original one. Specifically, it allows the objects to have arbitrary capacities, e.g., there could be a certain number of identical objects. The second one allows the situation where each query could have a *de-normalized* set of weights, which means different users have different priorities.

6.3 Facility Location Allocation Problem

The *facility location allocation* problem has been widely studied in the operational research community [47, 48]. Specifically, let P be a set of facilities and O be a set of users. Given a set of *matching strategies* and *matching constraints*, the objective of the *facility location allocation* is to determine a set of locations for the service-providers (whose locations are not fixed at the beginning) such that the

matching between P and O based on the given matching strategies and matching constraints is *optimized* (according to some appropriate criteria).

The solutions of the facility location allocation problem mainly have two steps: designing models to simulate the problem and developing algorithms based on the applied models. Among the existing models, *P-median model* [49], *covering model* [50] and *center model* [51] are three basic ones. Other models are usually built on these basic models by imposing appropriate constraints. As for algorithms, mainly two types of algorithms have been proposed. One is to utilize *mix-integer programming* (MIP) algorithms [52], and the other is heuristic-based [53].

In fact, the facility location allocation problem is very complicated. There are a lot of components that we need to specify, such as the matching strategies, matching constraints and optimization criteria. In fact, the matching problem is taken as a sub-component in the facility allocation problem. Specifically, after allocating locations to the facilities, we need perform the matching process based on the matching strategies and matching constraints to obtain the corresponding "optimal" matching, which could act as a measurement of the quality of the current location allocation.

7. Conclusion and Future Work

We make a conclusion of this survey in Section 7.1 and point out some future work directions in Section 7.2.

7.1 Conclusion

In this survey, we first study traditional matching problems in computer science and propose a general framework of the matching problems in spatial databases. Then, we introduce some spatial databases techniques (e.g., NN and RNN) that might be used for solving matching problems in spatial databases. After that, two well studied matching problems in spatial databases, namely SPM and CCA, are introduced in detail. Besides, we discuss some variants of the matching problems in spatial databases finally.

7.2 Future work

In this part, we point out several future directions related to matching problems in spatial databases.

7.2.1 Continuous Matching in Spatial Databases

Most existing matching problems assume *static* datasets. However, in many real-life scenarios (e.g., wireless network communication), the objects involved in the matching problems are *dynamic*. For instance, existing objects may leave the datasets, new objects might join the datasets, and the objects can also move in the data space . Thus, it is required that the matching algorithms should *continuously* report the *desirable* matchings according to changes in the dynamic environment. Still, much efforts need to be devoted to solving these issues.

7.2.2 Other Matching Strategies

Another research direction is motivated by the fact that up to now, only two basic matching strategies have been considered in the matching problems in spatial databases, namely the *fair assignment* (i.e., the one in SPM) and the *optimal assignment* (i.e., the one in CCA). That is, other traditional matching strategies (e.g., Leximin optimal assignment) have not been studied in the context of spatial databases yet. Thus, it would be quite interesting to explore some nice spatial properties that can be utilized for solving these matching problems in a more efficient way when they are put in the context of spatial databases.

7.2.3 Matching Problems in Other Types of Databases

It is promising to consider the matching problems in other types of databases, such as spatial network databases and relational databases.

Inspired by the idea of studying the matching problems in spatial databases, we attempt to consider the matching problems in other databases such as spatial network databases (SNDB) and relational databases (RDB) . Take the extension to SNDB as an example. Is is quite intuitive, since in many applications, the distance between two entities is not measured by their Euclidean distance studied in SPM and CCA, but their *network distance*.

REFERENCES

[1] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[2] D. Gale and L. Shapley. College admissions and the stability of marriage. *Amer. Math. Monthly*, 69:9–15, 1962.

[3] G. T. Ross and R. M. Soland. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8(1):91–103, 1975.

[4] Leong H. U, Man L. Yiu, Kyriakos Mouratidis, and Nikos Mamoulis. Capacity constrained assignment in spatial databases. In *SIGMOD*, 2008.

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[6] M. S. Hung. A polynomial simplex method for the assignment problem. *Operations research*, pages 595–600, 1983.

[7] ML Balinski. Signature methods for the assignment problem. *Operations research*, pages 527–536, 1985.

[8] D. P. Bertsekas. A new algorithm for the assignment problem. *Mathematical Programming*, 21(1):152–171, 1981.

[9] D. P. Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. *Annals of operations research*, 14(1):105–123, 1988.

[10] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1955):83–97, 1955.

[11] U. Derigs. A shortest augmenting path method for solving minimal perfect matching problems. *Networks*, 11(4):379–390, 1981.

[12] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM (JACM)*, 38(4):815–853, 1991.

[13] A. V. Goldberg and R. Kennedy. An efficient cost scaling algorithm for the assignment problem. *Mathematical Programming*, 71(2):153–177, 1995.

[14] R. W. Irving, P. Leather, and D. Gusfield. An efficient algorithm for the "optimal" stable marriage. *Journal of the ACM (JACM)*, 34(3):532–543, 1987.

[15] R. W. Irving, T. Kavitha, K. Mehlhorn, D. Michail, and K. Paluch. Rank-maximal matchings. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 68–75. Society for Industrial and Applied Mathematics, 2004.

[16] D.J. Abraham, K. Cechlarova, D.F. Manlove, and K. Mehlhorn. Pareto optimality in house allocation problems. *Lecture notes in computer science*, 3341:3–15, 2004.

[17] K. Mehlhorn. Assigning papers to referees. *Automata, Languages and Programming*, pages 1–2, 2009.

[18] R. W. Irving and P. Leather. The complexity of counting stable marriages. *SIAM Journal on Computing*, 15:655, 1986.

[19] Raymond C.-W. Wong, Yufei Tao, Ada W.-C. Fu, and Xiaokui Xiao. On efficient spatial matching. September 2007.

[20] K. Mouratidis and N. Mamoulis. Continuous spatial assignment of moving users. *The VLDB Journal*, 19(2):141–160, 2010.

[21] M. De Berg, O. Cheong, and M. Van Kreveld. *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc, 2008.

[22] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1993.

[23] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1196–1202. ACM, 2006.

[24] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Acm Sigmod Record*, volume 24, pages 71–79. ACM, 1995.

[25] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, 24(2):265–318, 1999.

[26] HV Jagadish, B. C. Ooi, K. L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b -tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems-TODS*, 30(2):364–397, 2005.

[27] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM, 1990.

[28] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD Record*, volume 29, pages 201–212. ACM, 2000.

[29] I. Stanoi, M. Riedewald, D. Agrawal, and A. El Abbadi. Discovery of influence sets in frequently updated databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 99–108, 2001.

[30] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 806–815. IEEE, 2007.

[31] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proceedings of the 31st international conference on Very large data bases*, pages 946–957. VLDB Endowment, 2005.

[32] E. Achtert, C. Bohm, P. Kroger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 515–526. ACM, 2006.

[33] K. Gowda and G. Krishna. The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (corresp.). *Information Theory, IEEE Transactions on*, 25(4):488–490, 1979.

[34] MR Brito, EL Chavez, AJ Quiroz, and JE Yukich. Connectivity of the mutual k-nearest-neighbor graph in clustering and outlier detection. *Statistics and Probability Letters*, 35(1):33–42, 1997.

[35] C. Ding and X. He. K-nearest-neighbor consistency in data clustering: incorporating local information into global optimization. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 584–589. ACM, 2004.

[36] K. C. Gowda and G. Krishnan. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, 10(2):105–112, 1978.

[37] W. Jin, A. Tung, J. Han, and W. Wang. Ranking outliers using symmetric neighborhood relationship. *Advances in Knowledge Discovery and Data Mining*, pages 577–593, 2006.

[38] C. Bohm and F. Krebs. High performance data mining using the nearest neighbor join. In *Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on*, pages 43–50. IEEE, 2002.

[39] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos. Closest pair queries in spatial databases. *ACM SIGMOD Record*, 29(2):189–200, 2000.

[40] G. R. Hjaltason and H. Samet. Incremental distance join algorithms for spatial databases. In *ACM SIGMOD Record*, volume 27, pages 237–248. ACM, 1998.

[41] C. Yang and K. I. Lin. An index structure for improving closest pairs and related join queries in spatial databases. In *Database Engineering and Applications Symposium, 2002. Proceedings. International*, pages 140–149. IEEE, 2002.

[42] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the eighth annual ACM symposium on Theory of computing*, pages 220–230. ACM, 1976.

[43] L. H. U, N. Mamoulis, and M. L. Yiu. Computation and monitoring of exclusive closest pairs. *IEEE Transactions on Knowledge and Data Engineering*, 20(12):1641–1654, 2008.

[44] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. 1993.

[45] L. H. U, N. Mamoulis, and K. Mouratidis. A fair assignment algorithm for multiple preference queries. In *VLDB*, 2009.

[46] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2005.

[47] S. H. Owen and M. S. Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423–447, 1998.

[48] P. Longley and M. Batty. *Advanced spatial analysis: the CASA book of GIS*. Esri Press, 2003.

[49] R. L. Church and C. S. ReVelle. Theoretical and computational links between the p-median, location set-covering, and the maximal covering location problem. *Geographical Analysis*, 8(4):406–415, 1976.

[50] J. A. Pacheco, S. Casado, J. F. Alegre, and A. Alvarez. Heuristic solutions for locating health resources. *Intelligent Systems, IEEE*, 23(1):57–63, 2008.

[51] M. S. Daskin. *Network and Discrete Location: Models Algorithms and Applications*. Wiley, Chichester, 1995.

[52] K. A. Willoughby. A mathematical programming analysis of public transit systems. *Omega*, 30(3):137–142, 2002.

[53] M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 127–137, 2001.