Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Matemáticas

Emparejamientos en gráficas bipartitas y problemas de asignación

Tesis que presenta

M. en C. Marcos César Vargas Magaña

Para obtener el grado de

DOCTOR EN CIENCIAS

En la Especialidad de Matemáticas

Director de la tesis:

Dr. Carlos Enrique Valencia Oleta

Ciudad de México Febrero, 2017

Center for Research and Advanced Studies
of the National Polytechnic Institute

Campus Zacatenco
Department of Mathematics

Matchings in bipartite graphs and assignment problems

Thesis presented by

M.Sc. Marcos César Vargas Magaña

to obtain the degree of

DOCTOR OF SCIENCE

in the speciality of Mathematics

Thesis Advisor:

Ph.D. Carlos Enrique Valencia Oleta

Ciudad de México Febrero, 2017

Resumen

El problema de asignación es uno de los problemas mas estudiado en optimización combinatoria, el cual ha sido estudiado por muchos autores a lo largo de muchos años; véase por ejemplo [**1**] y [**2**] y las referencias allí contenidas. La principal contribución de esta tesis es la introducción de algunas variantes del problema de asignación y la presentación de algunos algoritmos que permiten resolver estas variantes de manera eficiente.

Primeramente resolvemos algunas variantes con pesos a los problemas de encontrar todas las aristas que están en al menos un emparejamiento óptimo y la enumeración de todos los emparejamientos óptimos posibles. De manera mas precisa, dada una gráfica bipartita con pesos en las aristas, presentamos un algoritmo para encontrar todas las aristas que están en al menos un emparejamiento perfecto de peso mínimo con la misma complejidad en tiempo que resolver un problema de asignación en la misma gráfica. Actualmente el mejor tiempo es $O(m\sqrt{n}\log{(nW)})$, donde m es el número de aristas, n el número de vértices y W es el peso absoluto máximo. También resolvemos el problema de enumerar todos los emparejamientos perfectos de costo mínimo en tiempo $O(AP + \mathcal{M}\log n)$, donde AP es el tiempo para resolver el problema de asignación en la misma gráfica y $\mathcal{M}$ es el número de emparejamientos perfectos óptimos. Las versiones sin pesos de estos problemas han sido muy estudiadas por varios autores, véase por ejemplo [**20, 21, 22**] y [**23**]. Por último, originado de una aplicación a un problema real en la industria, proponemos y resolvemos una variante del problema de asignación donde tenemos un conjunto de aristas preferenciales, y el objetivo es encontrar un emparejamiento perfecto óptimo que contenga la mayor cantidad de aristas en este conjunto. Resolvemos este problema en la misma complejidad de tiempo del problema de asignación.

En una segunda parte de la tesis, estudiamos el problema de asignación aleatorio. Mas precisamente, estudiamos el problema de asignación con costos aleatorios con distribución exponencial en el caso de gráficas bipartitas no completas y bajo algunas hipótesis en la distribución de las aristas, obtenemos el valor esperado del peso mínimo de un emparejamiento de tamaño k. Otros autores han estudiado el caso para gráficas bipartitas completas.

Finalmente, en esta tesis presentamos una variante estocastica en dos fases del problema de asignación; el cual es parte de una colaboracion con científicos de los Laboratorios-HP y de la Universidad

de Missouri y el cual aun se encuentra en proceso. En este caso el proceso se realiza en dos fases, en el presente está la fase de asignación y en el futuro está la fase de elaboración. Cada vértice tiene una probabilidad conocida de no estar presente en la fase de elaboración. Si un vértice está presente y desocupado en la fase de elaboración, porque no fue asignado o su pareja no llegó a la fase de elaboración, este incurre en un alto costo de penalización. El objetivo es encontrar una asignación de cualquier tamaño que minimize el peso total de la asignación más el valor esperado de la penalización total. El algoritmo obtenido para resolver este problema tiene la misma complejidad de tiempo que resolver un problema de emparejamiento de peso mínimo en una gráfica del mismo tamaño. Este problema tiene muchas aplicaciones en la asignación de recursos, entre muchas otras.

Abstract

The assignment problem is one of the most studied problems in combinatorial optimization, has been studied by several authors through many years; see for instance [**1**] and [**2**] and the references therein. The contribution of this thesis is the introduction of some variants of the assignment problem and the presentation of algorithms to solve them in a efficient way.

Firstly, we solve the weighted variants of the problem of finding the edges that occur in at least one perfect matching and enumerating all the possible perfect matchings. More precisely, given a bipartite graph with edge weights, we give an algorithm for finding all the edges that occur in at least one minimum weight perfect matching in the same time complexity than solving the assignment problem in the same input. Currently the best time is $O(m\sqrt{n}\log{(nW)})$, where m is the number of edges, n is the number of vertices and W is the maximum absolute weight. We also solve the problem of enumerating all the minimum weight perfect matchings in time $O(AP + \mathcal{M}\log n)$, where AP is the time complexity for solving the assignment problem in the same input and $\mathcal{M}$ is the number of optimum perfect matchings. The unweighted variants of these problems have beed addressed by several researchers; see for instance [**20, 21, 22**] and [**23**]. Finally, originated from a real life application in the industry we propose and solve a variant of the assignment problem where we have a set of preference edges and the objective is to find an optimum perfect matching that contains as many of the preference edges as possible. We solve this problem in the same time complexity of the assignment problem.

In the second part of this thesis we study the random assignment problem. More precisely, we study the random assignment problem with random exponential weights in the case of non-complete bipartite graphs. Under some hypothesis on the distribution of the edges, we obtain the expected minimum weight of a matching of size k. Other authors have studied the case for complete bipartite graphs.

Finally, in this thesis we present an stochastic two phased version of the assignment problem, which is part of a work in progress in collaboration with scientists of the HP-Labs and the University of Missouri. The process is made in two phases, in the present is the assignment phase and in the future is the preforming phase. Each vertex has a known probability of not being present at the performing phase. If a vertex is present and idle at the performing phase, because it was not assigned or its mate did not reach the performing phase, it incurs in a high penalization cost. The objective is to find an assignment of any size that minimizes the total assignment weight plus the expected total penalization cost. The algorithm obtained to solve this problem is of the same time complexity than solving a minimum weight matching problem in a graph of the same size. This problem has many applications in the workforce allocation science, among many others.

Reconocimientos

El principal agradecimiento es para mi esposa Estela García, por acompañarme pacientemente a lo largo de todos estos sacrificios. También agradezco a mi madre Esperanza Magaña por su gran apoyo al inicio de mis estudios.

También quiero agradecer a mi asesor Carlos Valencia por su apoyo y orientación durante este proyecto y a mis sinodales.

A special thank to Cipriano Santos and Professor Haitao for allowing me the honor of working with they.

Quiero agradecer al Consejo Nacional de Ciencia y Tecnología (CONACYT) por otorgarme la beca durante estos años para poder concluir mis estudios.

Dedicatoria

Al gran “YO SOY”.

Dice el necio en su corazón: no hay Dios.
Salmos 14:1

Contents

Introduction

In the field of combinatorial optimization there are several problems that have been extensively studied. Here we work on one of the most important, the *Assignment Problem*. The importance of this problem relies on its wide field of practical applications and its theoretical results. The objective of this work is to explore, propose and solve several variations of the assignment problem.

The assignment problem is a fundamental theoretical problem and can be studied from several points of view. The most usual methods to attack this problem is by means of graph theory and linear optimization. In Chapter 1 we give all the necessary concepts of this topics to make this work as independent as possible.

Suppose that we have a set of n persons and a set of s jobs, where each person can perform a subset of the jobs. If a person is competent to perform a certain job, we say that they are a *feasible pair*. If we want a one-to-one assignment of as many persons as possible to perform one job each, then this problem is called the *Maximum cardinality Matching Problem*. The algorithms for solving this problem have reached impressive time bounds and currently the best known time is $O(m\sqrt{n})$ [**13**], where m is the number of feasible pairs. If the instance satisfies special properties, then there is an algorithm that solves it in $O(m)$ time [**15**]. This problem is highly related to the *Minimum Vertex Cover Problem*. One special case is when we ask for an assignment that covers all the persons and all the jobs. This problem is called *The Perfect Matching Problem*. We can also ask for all the edges that occur in at least one perfect matching. Or to enumerate all the perfect matchings. We discuss all these problems in detail in Chapter 2.

If each feasible pair person-job incurs in a cost that depends on the properties of the person and the job, then we can ask for a one-to-one assignment of all the persons to all the jobs that minimizes the total cost. This problem is called the *Minimum Weight Perfect Matching Problem* or *The Assignment Problem*. This problem and algorithms for solving it [**10, 7, 11**] are discussed in Chapter 3.

Moreover, we can ask for the corresponding weighted variants of problems involving unweighted perfect matchings. For example we can ask for all the edges that occur in at least one minimum weight perfect matching, or to enumerate all the minimum weight perfect matchings. Since there can be several perfect matchings of minimum weight, we can impose a set of preferred feasible pairs and ask for one optimum perfect matching that contains as many of our preferences as possible. All this novel problems and their solutions are some of our contributions. In Chapter 4 we discuss our results and the tools developed to solve this problems. These results are contained in [**32**].

What if the costs are random numbers on a continuous range. Can we say something relevant about the behavior of the minimum weight perfect assignments? The answer is yes. Several authors have proven important results about this problem [**31**] when the costs are uniform in $[0,1]$ or exponencial with rate 1. It has been proven that if any pair person-job is a feasible pair, then the expected value of a minimum weight perfect matching is $\sum_{i=1}^{n} 1/i^2$. This value clearly converges to $\pi^2/6$ as is proven by the Riemann zeta function $\zeta(2)$. There is even a result for the behavior of a minimum cost assignment of a given target size k. In Chapter 5 we describe the relevant results of this problem, and as another contribution we explore the case where not all pairs are feasible pairs. Under some hypothesis on the distribution of the random edges, we prove that the optimum cost for perfect assignments is $(1/d)\sum_{i=1}^{n} 1/i^2 \longrightarrow \zeta(2)/d$, where $d \in (0,1]$ is the density of the graph. We prove a result in general for assignments of a given target size k.

Now let us consider a variation where the costs are not random anymore. This time the assignment is made in the present and the the jobs are performed sometime in the future. Each person has a probability of not being available at the performing time and similarly each job has a probability of not being required at the performing time. If a person is present and idle at the performing time, it incurs a high penalization cost. And a job required at the performing time with no person to perform it also incurs in a penalization cost. The objective is to find an assignment of any size that minimizes the total assignment cost plus the expected total penalization cost. This is a problem with lots of applications in scheduling and workforce allocation problems. In Chapter 6 we show how to solve this problem efficiently.

Matching problems can be used to solve a wide range of interesting problems. Consider the following problem that arises in transportation science. We have a bus network with several trips that need to be served. The problem is to find the minimum number of vehicles needed to serve all the trips, along with the trips corresponding to each vehicle. To model this problem we can construct a network with directed edges. We have one node for each trip and one directed edge from a node i to a node j if trip j can be served by the same vehicle after serving trip i. Then the nodes in every directed path represents the trips that can be served by the same vehicle. Therefore, the solution to this problem is to find a minimum set of node-disjoint paths that cover all the nodes.

This problem can be transformed to a maximum cardinality matching problem. The bipartite graph induced by this transformation is as follows. We have as vertices a copy of the nodes in the left side and another copy in the right side. For every directed edge (i,j) we append an edge between node i in the left and node j in the right. It turns out that there is a one-to-one correspondence between the matchings and the node-disjoint paths. In particular a maximum cardinality matching leads to a minimum set of node-disjoint paths. This holds because every path in the directed network leaves two vertices of the bipartite graph unmatched, therefore minimizing the number of paths is equivalent to maximizing the cardinality of the matching.

The assignment problem enhances the reach of the bus network problem. For example, each edge can carry a cost representing how bad is the connection between a trip and the next. And the problem now is to find the minimum number of vehicles to serve all the trips at minimum cost. Using the previous reduction, this problem is reduced to the problem of finding a minimum weight maximum cardinality matching.

Another fields of applications occur in mathematics, computer science, operations research, biochemistry, electrical engineering, and many others. For example, solving the *All different constraint problem* [**25, 26**], computing the triangular form of sparse matrices [**27**], as an alternative for solving the shortest path problem [**10**], etc. More examples of interesting applications can be found in [**1**].

CHAPTER 1

Preliminaries

The objective of this chapter is to introduce the basic concepts and notations that we will use throughout the document. We want to give the necessary background related to topics in graph theory, linear optimization, optimization algorithms, probability and some other minor topics. If the reader is interested in more about this topics, please refer to the bibliography [**3, 1, 4, 2, 5**] and [**6**].

1. Graph theory

A *graph* G is the pair $G = (V, E)$, where V is a nonempty set of elements called *vertices* and E is a multi-set of unordered pairs of elements of V called *edges*. The set E is of the form $E = \{\{u, v\} : u, v \in V\}$. For simplicity we will denote an edge $\{u, v\}$ by the concatenation uv. We can represent a graph in the plane with points and lines, where each vertex $v \in V$ is represented by a point and each edge uv is represented by a line between point u and point v. This can be seen in figure 1.1.

Given an edge uv, it is said that the edge is *incident* to u and v and vice versa. It is also said that u and v are the *endpoints* of an edge uv. In a graph, a collection of edges that have the same endpoints are called *multiple edges* or *parallel edges*. An edge whose endpoint is the same vertex is called a *loop*. If a graph has no multiple edges nor loops then it is called a *simple graph*, otherwise is called a *multigraph*. We can see an example of this type of edges in figure 1.1.

Two vertices are said to be *adjacent* if there is an edge between them. Two edges are *adjacent* if they share one of its endpoints. A subset of edges such that no two of them are adjacent is called *independent*.

We denote the *cardinality* of a set A by $|A|$. A set A is said to be *maximal* with respect to some property $PROP$ if A has property $PROP$ and it is not a proper subset of any set that satisfies property $PROP$. The set A is said to be *maximum* with respect to property $PROP$ if A has property $PROP$ and any other set B having property $PROP$ is such that $|B| \leq |A|$.

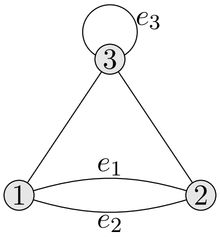


Figure 1.1. An example of a graph with two parallel Edges e_1, e_2; and a loop e_3.

Let $G = (V, E)$ and $G' = (V', E')$ be graphs. If $V' \subseteq V$ and $E' \subseteq E$, then G' is a *subgraph* of G, and G is a *supergraph* of G'. We denote this relation by $G' \subseteq G$. If $G' \neq G$ then the relations can be refered as *proper subgraph* and *proper supergraph* respectively and is denoted by $G' \subset G$.

An *induced subgraph* of $G = (V, E)$ is a subgraph $G' = (V', E')$ where E' is the subset of all the edges of G whose both endpoints are in V'. It is also said that G' is *induced* by V'.

The number of edges incident to a vertex v of G is called the *degree* of v. We denote the degree by $deg_G(v)$ or simply by $deg(v)$ if the graph G is understood. A vertex of degree zero is called *isolated*.

Given a subset $U \subseteq V$, the *set of neighbors* of U denoted by $N_G(U)$ (or simply $N(U)$) is composed by all the vertices not in U that can be reached from U by one single edge, it is defined by

$$N(U) = \{w \in V \backslash U \mid vw \in E \text{ for some } v \in U\}.$$

The *minimum degree* of a graph G is denoted by $\delta(G)$ and is defined by

$$\delta(G) = \min\{deg(v)| \ v \in V\}.$$

Similarly, the *maximum degree* $\Delta(G)$ of a graph G is

$$\Delta(G) = \max\{deg(v)| \ v \in V\}.$$

A *walk* in a graph is denoted by $W = v_0v_1v_2 \ldots v_k$ which is a subgraph on the vertices $\{v_0, \ldots, v_k\}$ where there is an edge between each consecutive pair v_i, v_{i+1}. Note that we can walk on an edge more than once. Vertices v_0 and v_k are the *ends* of the walk, all other vertices are the *inner* vertices of the walk. The *length* $len(W)$ of the walk is k. See figure 1.2.

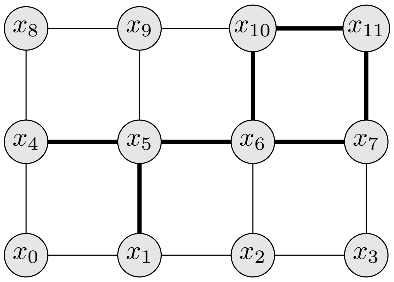


Figure 1.2. A walk in thick edges $W = x_4x_5x_6x_7x_{11}x_{10}x_6x_5x_1$.

Similar, a *trail* T is a walk with all of its edges distinct (see figure 1.3). Note that we can still repeat a vertex more than once. A *path* P is a trail with all of its vertices distinct (see figure 1.4). In the later case we have a path from vertex v_0 to vertex v_k.

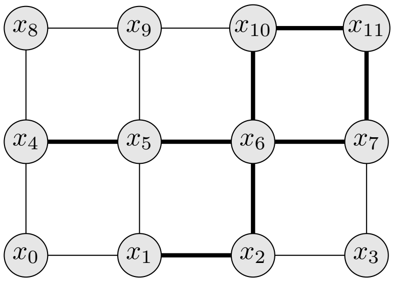


Figure 1.3. A trail in thick edges $T = x_4x_5x_6x_7x_{11}x_{10}x_6x_2x_1$.

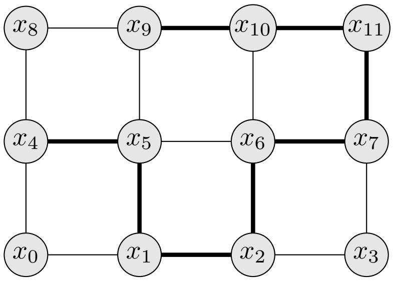


Figure 1.4. A path in thick edges $P = x_4x_5x_1x_2x_6x_7x_{11}x_{10}x_9$.

Given a path P and u, v two vertices in P, we write $P[u, v]$ to denote the *subpath* of P having endpoints u and v. Given A and B subsets of vertices, an *A-B-path* is a path $P = v_0v_1 \dots v_k$ such that $P \cap A = \{v_0\}$ and $P \cap B = \{v_k\}$. A *shortest path* from u to v is a path of minimum length with endpoints u, v.

The *distance* from a vertex u to a vertex v, denoted by $dist(u, v)$, is the length of a shortest path joining such vertices. If no such path exists we define $dist(u, v) = \infty$, otherwise

$$dist(u, v) = \min_{P \text{ is an } u-v-path} len(P). \tag{1}$$

The *diameter* of a graph $G = (V, E)$ is the largest distance between any two vertices of G. It is given by

$$diam(G) = \max_{u,v \in V} dist(u, v). \tag{2}$$

A *cycle* is a path that intersects at its endpoints. See figure 1.5. A graph with no cycles is called *acyclic*.

A non empty graph $G = (V, E)$ is said to be *connected* if for every $u, v \in V$ there is at least one u-v-path, otherwise it is called *disconnected*. A maximal connected subgraph of G is called a *connected component* of G (see figure 1.6).

A *k-regular* graph is a graph where $deg(v) = k$ for all $v \in V$. In particular, if the graph is 3-regular then it is called *cubic*. See figure 1.7.

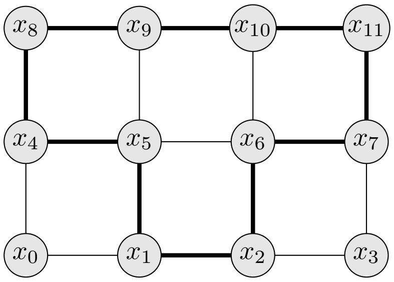


Figure 1.5. A cycle in thick edges $C = x_4x_5x_1x_2x_6x_7x_{11}x_{10}x_9x_8x_4$.

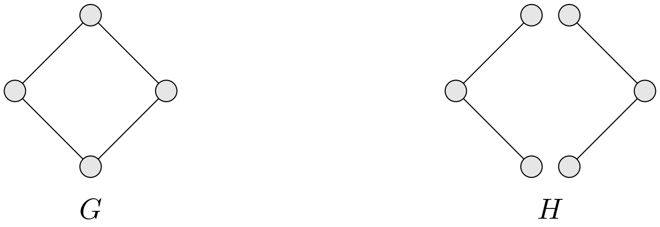
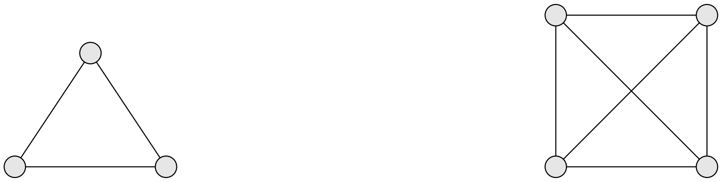


Figure 1.6. G a connected graph. H a disconnected graph with two components.



A 2-regular graph.

A cubic (3-regular) graph.

Figure 1.7. Examples of regular graphs.

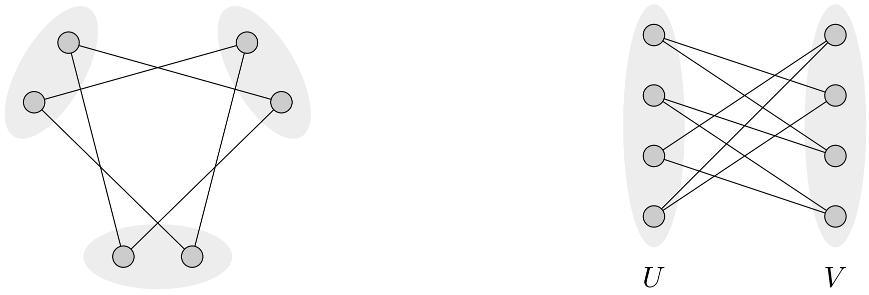
1.1. r-partite graphs

Let $G = (V, E)$ be a graph and $r \in \mathbb{N}$, $r \geq 2$. The graph G is called *r-partite* if its vertex set V can be partitioned into r disjoint classes $V = X_1 \sqcup X_2 \sqcup \cdots \sqcup X_r$ such that if $u, v \in X_i$, then u and v are not adjacent. In other words, every edge has its ends in different classes. See figure 1.8.

A very important case of r-partite graphs is for $r = 2$. In this case the graph is called *bipartite*. We denote a bipartite graph G by $G = (U \sqcup V, E)$ where U and V are the partitions of the vertices. A bipartite graph with $n = |U| = |V|$ is called *balanced*. Otherwise is called *unbalanced*. See figure 1.8.

One interesting result about the characterization of bipartite grpahs that can be found in [**5**] is the following.

Proposition 1.1 ([**5**]). A graph G is bipartite if and only if any cycle in G is of even length.

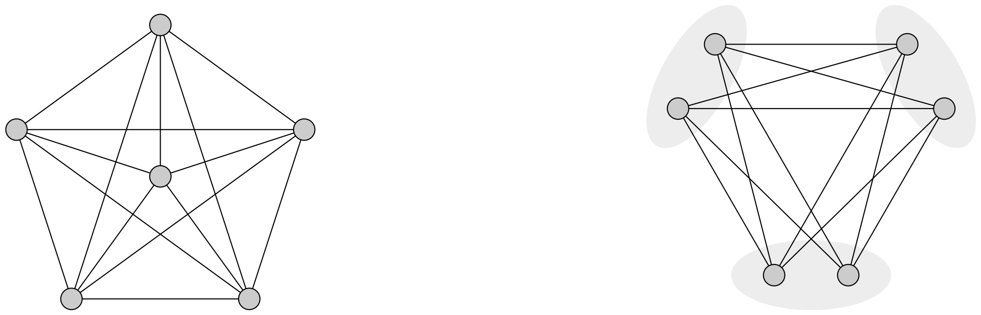


A 3–partite graph. A balanced biartite graph.

Figure 1.8. Examples of r-partite graphs.

1.2. Complete graphs

A *complete graph* is a simple graph where every $u \neq v \in V$ are adjacent. Such graphs are generally denoted by K_n, where n is the cardinality of the vertex set. A *complete r-partite* graph $K_{n_1,\ldots,n_r} = (X_1 \sqcup X_2 \sqcup \cdots \sqcup X_r, E)$, where $n_1 = |X_1|, \ldots, n_r = |X_r|$, is a simple graph where every pair of vertices $u \neq v$ not in the same partition are adjacent. In particular, a complete bipartite graph $G = (U \sqcup V, E)$ with $n = |U|$, $s = |V|$ is denoted by $K_{n,s}$. See figure 1.9.



The complete graph K_6. The complete 3-partite graph $K_{2,2,2}$.

Figure 1.9. Examples of complete graphs.

1.3. Trees and forests

A *tree* is a connected acyclic graph. A *forest* is a graph where every connected component is a tree. A *leaf* in a tree is a vertex of degree one. A *rooted tree* is a tree with a special vertex r called the *root*, see figure 1.10.

Respect to the characterization of trees, there is an interesting result given in the following Theorem, which can be found in [**5**].

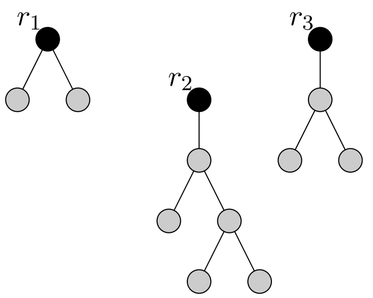


Figure 1.10. A forest composed by 3 trees with roots $\{r_1, r_2, r_3\}$.

Theorem 1.2. Let $T = (V, E)$ be a graph. The following statements are equivalent

(1) T is a tree.

(2) Any two vertices of T are linked by a unique path in T.

(3) T is minimally connected, i.e. T is connected but $T \backslash \{uv\}$ is disconnected for every edge $uv \in T$.

(4) T is maximally acyclic, i.e. T contains no cycle but $T \cup \{uv\}$ does, for any two non-adjacent vertices $u, v \in T$.

A *spanning tree* of a connected graph G is a connected subgraph over all the vertices of G which is a tree. Note that every tree on n vertices has $n-1$ edges.

1.4. Digraphs

A *directed graph* (or *digraph*) $D = (V, A)$ consists of a set of vertices V and a set A that contains ordered pairs of vertices called *arcs* or *directed edges*. An arc from u to v is denoted (uv) and means that we have an edge directed from u to v. In this case the vertex u is the *tail* and the vertex v is the *head*. Arcs can be represented by arrows, see figure 1.11. The *reverse arc* of an arc (uv) is the arc given by $(uv)^R = (vu)$.

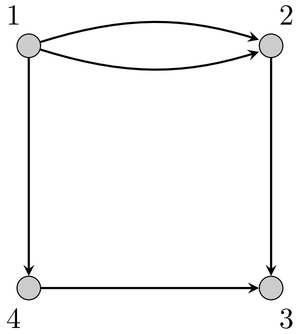


Figure 1.11. A digraph.

Definitions for *walk*, *trail*, *path*, *cycle*, etc. can be easily extended to digraphs. The only difference is that we must respect the direction of the arcs. Arcs can be *parallel* only if they have the same orientation. We also have the following definitions for digraphs.

- $deg^{-}(v) = |\{(uv) \in E\}|$, the *indegree*.
- $deg^{+}(v) = |\{(vu) \in E\}|$, the *outdegree*.
- $N^{-}(U) = \{v \in V \backslash U \mid (vu) \in E$ for some $u \in U\}$.
- $N^{+}(U) = \{v \in V \backslash U \mid (uv) \in E$ for some $u \in U\}$.
- $\delta^{-}(D) = \min\{deg^{-}(v) \mid v \in V\}$.
- $\delta^{+}(D) = \min\{deg^{+}(v) \mid v \in V\}$.
- $\Delta^{-}(D) = \max\{deg^{-}(v) \mid v \in V\}$.
- $\Delta^{+}(D) = \max\{deg^{+}(v) \mid v \in V\}$.

A digraph is *strognly connected* if we can go from any vertex to any other vertex using a directed path. A *strongly connected component* of a digraph is a maximal strongly connected subgraph. Note that the strongly connectec components of a digraph are disjoint, because if two of them share one vertex, then the union of the two components is a strongly connected component, which violates the maximality condition of the components.

Proposition 1.3. In a digraph, an arc is part of a directed cycle if and only if the arc is part of a strongly connected component.

Proof. If we contract every strongly connected component to a single vertex the resulting digraph is acyclic, and therefore any arc not in a strongly connected component cannot be in a directed cycle. Then any arc in a directed cycle is in a strongly connected component.

If an arc (uv) is in a strongly connected component, then there exists a directed path P from u to v and the path $P \cup \{(uv)\}$ is a directed cycle. □

1.5. Matchings

Our interest will focus on bipartite graphs, therefore we only consider matchings in bipartite graphs. Matchings in general graphs have been well studied but they are out of the scope of this work.

Let $G = (U \sqcup V, E)$ be a bipartite graph. A *matching* is given by a subset $M \subseteq E$ of independent edges. A vertex v is said to be *covered* or *assigned* under the matching M if v is incident to some edge of the matching. Otherwise the vertex is called *uncovered* or *unassigned*.

An *integer weight function* over the edges of a graph is given by $w : E \to \mathbb{Z}$ and assigns one integer weight to each edge. We can define the *weight* of a matching M as

$$w(M) = \sum_{uv \in M} w(uv). \tag{3}$$

The *cardinality* of a matching is the number of edges it has. A *perfect matching* in a balanced bipartite graph is a matching that covers all the vertices. A *one-side perfect matching* in an unbalanced graph is a matching that covers all the vertices of the smallest side.

An *alternating path* respect to a matching is a path that alternates between edges in the matching and edges not in the matching. An *augmenting path* respect to a matching is an alternating path with endpoints uncovered by the matching. Augmenting paths have the property that we can use the symmetric difference $M \triangle P$ between the matching M and the path P to increase the cardinality of the matching by one. The *symmetric difference* of two sets A, B is defined as $A \triangle B = (A \backslash B) \cup (B \backslash A)$.

There is an interesting result about k-regular bipartite graphs, that states that every regular bipartite graph has perfect matchings.

Proposition 1.4 ([**1**], *Corollary 2.3*). Every k-regular bipartite graph, with $k \geq 1$, has a perfect matching.

2. Linear optimization

In the theory of linear optimization the objective is to use linear inequalities to optimize a restricted linear objective function.

A linear optimization problem looks like the following *general linear programming problem*.

$$\begin{aligned} \text{minimize:} \quad & c \cdot x \\ \text{subject to:} \quad & a_i \cdot x = b_i, \quad i \in I_1, \\ & a_i \cdot x \geq b_i, \quad i \in I_2, \\ & a_i \cdot x \leq b_i, \quad i \in I_3, \\ & x_j \geq 0, \qquad j \in N_1, \\ & x_j \leq 0, \qquad j \in N_2. \end{aligned} \tag{4}$$

Where $c = (c_1, \dots, c_n)$ is the *cost vector*, $x = (x_1, \dots, x_n)$ is the set of *decision variables*, and the a_i are n-dimensional vectors. I_1, I_2 and I_3 are disjoint subsets of indices for the linear constraints. And N_1, N_2 are disjoint subsets of $\{1, \dots, n\}$ for the sign of the desition variables. Variables with index not in $N_1 \cup N_2$ are sign free.

A vector x that satisfies all of the constraints is called a *feasible solution*. The set of all feasible solutions is called the *feasible region*. The linear function $c \cdot x = \sum_{j=1}^{n} c_j x_j$ is called the *objective function*. A feasible solution that minimizes the objective function is called an *optimal solution*, and the respective value is called the *optimal value*. If there is no bound below on the values induced by the feasible solutions then we say that the optimal value is $-\infty$. Note that if we want to maximize $c \cdot x$, then we just need to minimize $-c \cdot x$.

Any constraint of the form $a_i \cdot x = b_i$ is equivalent to the two constraints $a_i \cdot x \leq b_i$ and $a_i \cdot x \geq b_i$. And every constraint of the form $a_i \cdot x \leq b_i$ is equivalent to $-a_i \cdot x \geq -b_i$. Therefore, any linear program can be expresed in the following form.

$$\begin{aligned} \text{minimize:} \quad & c \cdot x \\ \text{subject to:} \quad & a_1 \cdot x \geq b_1 \\ & a_2 \cdot x \geq b_2 \\ & \vdots \\ & a_m \cdot x \geq b_m \end{aligned} \tag{5}$$

Which in turn can be expresed equivalently as

$$\begin{aligned} \text{minimize:} \quad & c \cdot x \\ \text{subject to:} \quad & Ax \geq b \end{aligned} \tag{6}$$

Where A is the matrix with rows $a_1, \ldots, a_m$, and $b = (b_1, \ldots, b_m)$ is the *requirement vector*.

The general linear program (6), and therefore any linear program, can be transformed into the following *standard form linear program*:

$$\begin{aligned} \text{minimize:} \quad & c \cdot x \\ \text{subject to:} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{7}$$

This can be accomplished via a process described in [**3**, Chapter 1], which consists of elimination of free variables and elimination of inequality constraints. The two resulting problems are equivalent in the sense that given a feasible solution to one problem, we can construct a feasible solution of the other with the same value. This implies that both problems have the same optimal value.

Linear programming techniques are extremely useful in a wide range of optimization problems. Many interesting optimization problems can be translated to the language of linear optimization even though in many cases this is not obvious. In the following example we show an interesting problem taken from [**3**, exercise 1.12]. Our solution to this exercise is based on some basic properties of the distance from a point to a multidimensional plane.

Example 1.5. Consider a set P described by linear inequality constraints, that is, $P = \{y \in \mathbb{R}^n : a_i \cdot y \leq b_i,\ i = 1, \ldots, m\}$. A ball with center x and radius r is defined as the set of all points within euclidean distance r from x. We are interested in finding a ball with the largest posible radius, which is entirely contained within the set P. The center of such ball is called the *Chebychev center* of P. Provide a linear formulation of this problem.

Solution. Observe that the largest ball should touch at least one of the hyperplanes $a_i \cdot y = b_i$, and the segment from the center x to such plane is orthogonal to the plane. Therefore we need to maximize the distance from the candidate center x to the closest hyperplane.

It turns out that the distance from a point x to the i-th plane $a_i \cdot y = b_i$ is given by $dist_i(x) = \dfrac{-a_i \cdot x + b_i}{|a_i|}$. One important observation is that the sign of $dist_i(x)$ is positive if and only if the point x is in the side of the plane oposite to where the vector a_i is pointing to. And this is always the case in the feasible region P. This ensures that $dist_i(x) \geq 0\ \forall x \in P$.

This lead us to the following formulation.

$$\begin{array}{ll} \text{maximize:} & \min_i \left\{ \dfrac{-a_i \cdot x + b_i}{|a_i|} \right\} \\ \\ \text{subject to:} & a_i \cdot x \leq b_i \qquad \text{i=1,}\ldots\text{, m.} \end{array}$$

Note that the optimum objective value is the radius of the optimum ball, because each distance from the point to a plane is a posible radius. Since we need the minimum radius we can reformulate as:

$$\begin{array}{rlll} \text{maximize:} & \text{r} & & \\ \text{subject to:} & a_i \cdot x + |a_i| r & \leq b_i & \text{i=1,}\ldots\text{, m} \\ & a_i \cdot x & \leq b_i & \text{i=1,}\ldots\text{, m.} \end{array}$$

The resulting x is the center and r is the radius of the optimum ball. □

A *polyhedron* is a set of the form $\{x \in \mathbb{R}^n : Ax \geq b\}$, where A is an $m \times n$ matrix and b is an m-dimensional vector. As we can observe, any feasible region of a linear program is a polyhedron. A vector $x \in P$ is a *vertex* of P if there exists some n-dimensional vector c such that $c \cdot x < c \cdot y$ for all $x \neq y \in P$. In other words, x is a vertex if there is an hyperplane that touches the polyhedron only at point x. In particular, the polyhedron described by a linear program in standar form is called a *polyhedron in standard form*. The following corollary found in [**3**] explains about the existence of vertices in polyhedra.

Corollary 1.6 ([**3**], *Corollary 2.2*). Every nonempty bounded polyhedron and every nonempty polyhedron in standard form has at least one vertex.

One of the most important results of linear optimization is given in the following theorem, which raises the importance of the vertices of a polyhedron.

Theorem 1.7 ([**3**], *Theorem 2.8*). Consider the linear programming problem of minimizing $c \cdot x$ over a polyhedron P. Suppose that P has at least one vertex. Then either the optimal value is equal to $-\infty$, or there exists a vertex which is optimal.

The importance of this theorem relies on the fact that the efficient algorithms that solve linear programming problems like the simplex method ([**3**], Chapter 3), exploit this property at each iteration to go from one vertex to another one of smaller cost. Since we always have a vertex that has optimum value then the algorithm will reach an optimum solution at some point.

If a vector x satisfies with equality a linear equality or inequality constraint, then it is said that the constraint is *active* at x. Given a polyhedron described by linear equality and inequality constraints, a vector x is a *basic solution* if (i) all equality constraints are active and (ii) out of all the constraints that are active, n of them are linearly independent. Moreover, the vertex x is called a *basic feasible solution* if it is a basic solution and satisfies all of the constraints. Indeed it is proven in [**3**] that x is a vertex if and only if it is a basic feasible solution.

The following theorem shows a more algebraic point of view of the vertices of polyhedra in standard form. It basically states that every vertex can be obtained by means of solving a system of linear equations.

Theorem 1.8 ([**3**], *Theorem 2.4*). Consider the constraints $Ax = b$ and $x \geq 0$ and assume that the $m \times n$ matrix A has linearly independent rows. A vector $x \in \mathbb{R}^n$ is a basic solution if and only if $Ax = b$ and there exists a subset of indices $B(1), \ldots, B(m)$ such that:

(a) The columns $A_{B(1)}, \ldots, A_{B(m)}$ are linearly independent.

(b) If $i \neq B(1), \ldots, B(m)$, then $x_i = 0$.

Since every vertex is a basic solution, then every vertex can be obtained by solving a system of linear equations of the form $Bx = b$, where B is an $m \times m$ submatrix of A with columns $A_{B(1)}, \ldots, A_{B(m)}$.

In the following corollary is given an important result about the number of vertices of a linear programming problem.

Corollary 1.9 ([**3**], *Corollary 2.1*). Given a finite number of linear inequality constraints, there can only be a finite number of vertices in the feasible region.

The proof of this corollary is based on the fact that every subset of n linearly independent constraints define a unique basic solution, therefore the number of vertices is bounded by the number of ways that we can chose n constraints out of the total finite number of constraints. This result is important because this implies that any algorithm that searches optimal solutions in the vertices of the polyhedron will terminate after a finite number of steps. However, the number of vertices can be exponential in the number of constraints. The most famous example of this is the unit cube $\{x \in \mathbb{R}^n : 0 \leq x_i \leq 1\}$ which is defined by $2n$ constraints but has 2^n vertices.

2.1. Duality theory

Now we present an alternative method for attacking linear programming problems. This approach is known as *duality theory*. Given a linear program called the *primal problem*, the objective of duality is to formulate a different linear program known as the *dual problem*, which is based on the primal problem.

The basic idea behind duality theory is to assign a price variable with each constraint and search for optimal prices under which the presence or absence of the constraints does not affect the optimal value. In general terms, in the dual problem there is a *dual price variable* for each primal constraint and a dual constraint for each primal variable. We give a precise relation in the following formulations, where the matrix A of the primal problem has rows a_i and colums A_j. We have the decision variables $x = (x_1, \ldots, x_n)$, the cost vector $c = (c_1, \ldots, c_n)$, the dual prices $p = (p_1, \ldots, p_m)$ and the requirement vector $b = (b_1, \ldots, b_m)$.

$$
\begin{array}{rlll@{\qquad\qquad}rlll}
 & \textbf{Primal problem} & & & & \textbf{Dual problem} & & \\
\text{minimize:} & c\cdot x & & & \text{maximize:} & p\cdot b & & \\
\text{subject to:} & a_i\cdot x\geq b_i, & i\in M_1, & & \text{subject to:} & p_i\geq 0, & i\in M_1, & \\
 & a_i\cdot x\leq b_i, & i\in M_2, & & & p_i\leq 0, & i\in M_2, & \\
 & a_i\cdot x= b_i, & i\in M_3, & & & p_i \text{ free}, & i\in M_3, & \\
 & x_j\geq 0, & j\in N_1, & & & p\cdot A_j\leq c_j, & j\in N_1, & \\
 & x_j\leq 0, & j\in N_2, & & & p\cdot A_j\geq c_j, & j\in N_2, & \\
 & x_j \text{ free}, & j\in N_3, & & & p\cdot A_j= c_j, & j\in N_3. &
\end{array} \tag{8}
$$

Observe that while in the primal problem we address a minimization problem, in the dual problem we address a maximization problem. This makes sense because of the fact that $p \cdot b \leq c \cdot x$ for every feasible solutions x and p, as we will see later. In particular, for feasible solutions we have that $p \cdot b = c \cdot x$ if and only if both solutions are optimal.

If we focus on standard from problems, we get the following primal dual relation, where p' is the vector p transposed.

$$
\begin{array}{rl@{\qquad\qquad}rl}
 & \textbf{Primal problem} & & \textbf{Dual problem} \\
\text{minimize:} & c\cdot x & \text{maximize:} & p\cdot b \\
\text{subject to:} & Ax=b, & \text{subject to:} & p'A\leq c. \\
 & x\geq 0, & &
\end{array} \tag{9}
$$

The following result found in [**3**] states, in informal language, that the dual of the dual is the primal.

Theorem 1.10 ([**3**], *Theorem 4.1*). If we transform the dual of a primal problem into an equivalent minimization problem and then form its dual, then we obtain a problem equivalent to the original problem.

The following results which can be found in [**3**] are the central foundations of duality theory.

Theorem 1.11 ([**3**], *Theorem 4.3, Weak duality*). If x is a feasible solution of the primal problem and p is a feasible solution of the dual problem, then $p \cdot b \leq c \cdot x$.

Corollary 1.12 ([**3**], *Corollary 4.2*). Let x and p be feasible solutions to the primal and the dual, and suppose that $p \cdot b = c \cdot x$. Then, x and p are optimal solutions to the primal and the dual respectively.

Theorem 1.13 ([**3**], *Theorem 4.4, Strong duality*). If a linear programming problem has an optimal solution, so does its dual, and the respective optimal values are equal.

There is a pair of necessary and sufficient conditions for the feasible primal and dual solutions to be optimal. They are given in the following theorem that can be found in [**3**].

Theorem 1.14 ([**3**], *Theorem 4.5, Complementary slackness*). Let x and p be feasible solutions to the primal and the dual problems respectively. The vectors x and p are optimal solutions for the two respective problems if and only if:

$$p_i(a_i \cdot x - b_i) = 0, \qquad \forall i, \tag{10}$$

$$(c_j - p \cdot A_j)x_j = 0, \qquad \forall j. \tag{11}$$

3. Probability

In probability theory, a random variable with *Bernoulli distribution* takes value 1 with success probability $p \in [0,1]$ and value 0 with failure probability $(1-p)$. This distribution is useful for modeling scenarios where we only get two posible outcomes. The value 1 represents one of the outcomes, usually the success, and the value 0 represents the other outcome, the failure.

A *Bernoulli random variable* is a variable which takes random values according to the bernoulli distribution, that is, if X is such random variable with parameter p, then $Pr(X = 1) = p$ and $Pr(X = 0) = 1 - p$. The *probability mass function* f of the bernoulli distribution is given by

$$f(k,p) = \begin{cases} p & \text{if } k = 1, \\ 1-p & \text{if } k = 0. \end{cases} \tag{12}$$

The *expected value* of a random variable X is like a probabilistic average. Formally, if a random variable takes values in the set $\{x_1, \ldots, x_n\}$. Then the expected value of the random variable is:

$$E[X] = x_1 \cdot Pr(X = x_1) + \cdots + x_n \cdot Pr(X = x_n). \tag{13}$$

From the definition of expected value, we can get the expected value of a bernoulli random variable with parameter p as follows.

$$E[X] = 0 \cdot Pr(X = 0) + 1 \cdot Pr(X = 1) = p. \tag{14}$$

The *variance* of a random variable measures how far a set of values of the variable is spread. The lower the variance the more concentrated the values are at the mean. In particular, a variance of zero indicates that all the values are identical. Formally, the variance of a random variable X is defined as

$$Var(X) = E[(X - E[X]^2)] = E[X^2] - (E[X])^2. \tag{15}$$

From the definition, the variance of a bernoulli variable is

$$Var(X) = p - p^2 = p(1-p). \tag{16}$$

As an observation, the maximum of the variance expression $p(1-p)$ is reached at $p = 1/2$. Therefore, the maximum variance that a bernoulli distribution can have is 0.25.

CHAPTER 2

Bipartite matching problems

In this chapter we talk about matching problems. These type of problems are related to the search of matchings in bipartite graphs with no weights. The objective of this chapter is to introduce some of the most important matching problems as well as their algorithms, to prepare the way for the development of our algorithms of Chapter 4. The importance of these algorithms relies on the fact that we will address some problems for weighted matchings in bipartite graphs and we will attack them via a reduction to the respective unweighted problems.

From the point of view of graph theory, the *bipartite perfect matching problem* consists of finding a perfect matching in a bipartite graph $G = (U \sqcup V, E)$. Along with this problem, there are other problems of interest in matching problems. For example, we have the *bipartite maximum matching problem* which asks for a maximum cardinality matching. The *bipartite maximal matching problem* which asks for a maximal cardinality matching. Observe that every maximum matching is maximal, but a maximal matching may not be maximum. In this chapter we will only talk about the bipartite perfect matching problem, which we will refer as the perfect matching problem. Observe that every perfect matching is a maximum matching and a maximal matching.

For this chapter we will reserve the symbols $m = |E|$, $n = |U| = |V|$ for the case of balanced graphs, and $n = |U| \leq |V| = s$ for the case of unbalanced graphs. If we want to use this symbols for another temporal purposes we will let it clear for the reader.

The perfect matching problem can be interpreted as follows. We have n men and n women. Each men has a subset of friends in the women set. The objective is to marry all the men to all the women such that every marriage is composed by a men and one of his women friends.

Given a balanced bipartite grpah $G = (U \sqcup V, E)$, its *adjacency matrix* is an $n \times n$ matrix A with entries a_{uv} given by

$$a_{uv} = \begin{cases} 1 & \text{if } uv \in E, \\ 0 & \text{if } uv \notin E. \end{cases} \tag{17}$$

Note that each non-zero entry corresponds to an edge in the graph. Another way of seeing the perfect matching problem is as the problem of selecting n entries of value 1 in the adjacency matrix, such that there is one entry in every row and in every column.

The following theorem gives a necessary and sufficient condition for the existence of perfect matchings in bipartite graphs. This theorem is known as the *marriage theorem*.

Theorem 2.1 (*Marriage theorem,* [**1**] *Theorem 2.2*). Let $G = (U \sqcup V, E)$ be a bipartite graph with $|U| = |V|$. There exists a perfect matching (marriage) in G if and only if G fulfills the following *Hall's condition*.

$$|U'| \leq |N(U')|, \qquad \text{for every subset } U' \subseteq U. \tag{18}$$

As a consequence of the Theorem 2.1, it is not hard to prove that every k-regular bipartite graph has a perfect matching, as we saw in the Proposition 1.4. This holds because for all U', the subgraph induced by $U' \sqcup N(U')$ satisfies $k|U'| = \sum_{v \in N(U')} deg(v) \leq k|N(U')|$, threfore $|U'| \leq |N(U')|$.

A *vertex cover* C in a bipartite graph $G = (U \sqcup V, E)$ is a subset of vertices such that every edge of G is adjacent to some vertex in C. There is an interesting problem related to vertex covers that asks for a vertex cover of minimum cardinality. We mention this problem because there is a strong relation between vertex covers and maximum cardinality matchings.

Theorem 2.2 (*König's matching theorem,* [**1**] *Theorem 2.7*). In a bipartite graph the cardinality of a minimum vertex cover equals the cardinality of a maximum matching. That is

$$\min_{C \text{ vertex cover}} |C| = \max_{M \text{ matching}} |M|.$$

Observe that in the case of a bipartite graph with perfect matchings, a minimum vertex cover is given by any of U or V. Also note that a vertex cover can be seen as a subset of rows and columns of the adjacency matrix that cover all the 1-entries.

1. Maximum cardinality matchings

There are several algorithms for finding maximum matchings in bipartite graphs. The classic method for solving this problem is based on labeling techniques, and has $O(mn)$ time complexity. We will present the algorithm later. The most important method for finding maximum matchings is due to Hopcroft and Karp [**13**], and is based on a technique that at each iteration increments the size of the current matching using a maximal set of augmenting paths. This algorithm achieves the best known time complexity of $O(m\sqrt{n})$.

A bipartite graph is called *convex* if for every $j < k$, the presence of edges $u_i v_j$ and $u_i v_k$ implies the presence of edges $u_i v_h$ for all $j < h \leq k$. As we will see later, in this particular case, a maximum cardinality matching can be found in $O(n)$ time in an efficient implementation of the Glover's algorithm [**15**].

Lemma 2.3 (*Augmentation lemma,* [**1**] *Lemma 3.3*). If M is not a maximum matching in G, then there exists an augmenting path P with respect to M such that $M' = M \triangle P$ is a matching in G of cardinality $|M| + 1$.

The following corollary gives a rule that helps us to know when we have reached a maximum matching in a sequence of augmentations.

Corollary 2.4. A matching M has maximum cardinality if and only if there is no augmenting path respecto to M.

The following algorithm, which can be found in [**1**, Algorithm 3.1], shows the classic method for finding maximum matchings via a labeling technique. The set L contains vertices of U and the set R collects vertices of V.

Algorithm 2.5 (Classic maximum cardinality matching algorithm)**.**

```
Input: Bipartite graph $G = (U \sqcup V, E)$.
Output: A maximum cardinality matching $M$.

Procedure maximum_matching($G$)
    let $M$ be a matching, possibly empty;
    let $L$ contain all unmatched vertices of $U$;
    $R = \phi$;
    while($L \cup R \neq \phi$) do
        chose a vertex $x$ from $L \cup R$;
        if($x \in L$) Scan_leftvertex($x$) else Scan_rightvertex($x$);
    end;
    return $M$;
end

Procedure Scan_leftvertex($u$)
    $L = L \backslash \{u\}$;
    for(all unlabeled $v \in N(u)$) do
        label $v$ as $l(v) = u$;
        $R = R \cup \{v\}$;
    end;
end
```

```
Procedure Scan_rightvertex($v$)
    $R = R\backslash\{v\}$;
    if(there is a matching edge $uv \in M$);
        label $u$ as $r(u) = v$;
        $L = L \cup \{u\}$;
    else [we have found an augmenting path]
        find the augmenting path by backtracking the labels: $P = (\dots, r(l(x)), l(x), x)$;
        $M = M \triangle P$;
        let $L$ contain all unmatched vertices of $U$;
        $R = \phi$;
        cancel all labels;
    end;
end
```

The running time of the algorithm is derived from the fact that at each augmentation the matching increases its cardinality by one. Therefore we need at most n augmentations. Since every vertex is labeled at most once per augmentation, then we don't scan the neighbors of a vertex $u \in U$ more than once, therefore we make an augmentation in $O(m)$ time. This gives a total $O(mn)$ time.

Since we can start the algorithm with an arbitrary matching, the best option is to start with the largest matching that we can find using greedy-like methods. One important observation about this algorithm is that it also produces a vertex cover for the input bipartite graph. This vertex cover is given by the unlabeled vertices of U and the labeled vertices of V.

This is one of the simplest efficient algorithms for finding maximum cardinality matchings. As we mentioned before, the most famous algorithm is the Hopcroft-karp algorithm [**13**] with time complexity $O(m\sqrt{n})$ [**1**, Theorem 3.11], which can be consulted in [**13, 1**].

For reference, we present following the algorithm of Glover [**15**] for finding maximum cardinality matchings in convex bipartite graphs. This algorithm can be implemented in $O(n)$ time complexity using an efficient data structure developed by Gabow and Tarjan [**17**]. Remember that a bipartite graph $G = (U \sqcup V, E)$ is convex if we can arrange its vertex sets $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_s\}$ such that the set of neighbors of every $u \in U$ is of the form $N(u) = \{v_j, v_{j+1}, \dots, v_k\}$. Such arrangement can be found in $O(m + n)$ time [**16**]. Assuming this arrangement has been found, the pseudocode for the algorithm is the following.

Algorithm 2.6 (Maximum matchings in convex bipartite graphs, [**1**] Algorithm 3.4)**.**

Input: Convex bipartite graph $G = (U \sqcup V, E)$.
Output: A maximum cardinality matching M.

```
Procedure convex_maximum_matching($G$)
    $M = \phi$;
    for(every $u \in U$) do $\alpha(u) = \max\{j \mid v_j \in N(u)\}$;
    for($j = 1$ to $s$) do
        if($v_j$ has unmatched neighbors)
            find an unmatched neighbor $u$ of $v_j$ with minimum value $\alpha(u)$;
            add the edge $uv_j$ to $M$;
        end;
    end;
end
```

There is a probabilistic algorithm for finding the cardinality of a maximum matching and the matched vertices without giving the matching. The result is summarized in the following theorem.

Theorem 2.7 ([**1**], *Theorem 3.22*). In a bipartite graph $G = (U \sqcup V, E)$, with $n = |U|$ and $s = |V|$, the cardinality of a maximum matching and the matched vertices can be found in $O(n^{\beta-1}s)$ arithmetic operations, where $O(n^{\beta})$ is the time complexity for multiplying two $n \times n$ matrices.

Currently, the best time complexity for multiplying two $n \times n$ matrices is $O(n^{2.376})$ [**18**]. For the particular case of perfect matchings, there is a probabilistic algorithm given by Lovász [**19**] that helps to determine the existence of a perfect matching in a balanced bipartite graph. At each try, the algorithm tries to determine whether the graph has a perfect matching or not. The algorithm returns the wrong answer with probability $1/m$. Therefore, after k iterations, the algorithm will return the correct answer with probability $(1 - 1/m^k)$. Since the algorithm mostly consists of computing the determinant of an $n \times n$ matrix at each iteration, then the time complexity for k iterations is $O(n^{\beta}k)$. The algorithm can be found in [**1**, Algorithm 3.5].

2. Finding the edges that occur in at least one perfect matching

Consider the following problem. There is a matchmaking agency that matches single men to single women. There are n men and n women. Given the set of preferences of each man and woman the agency can determine all the pairs man-woman that may like each other. Since the agency wants to match all men to all women in a one-to-one relation, then the agency does not want to present a woman to a man that can prevent a full assignment, because this is not good for the agency due to profit optimization. Therefore the agency wants to remove all the pairs man-woman that prevent a full assignment, and only present couples that allow a full assignment. This problem can be seen as a bipartite matching problem where we want find all the edges that occur in at least one perfect matching. The objective of this section is to present an algorithm that solves this problem efficiently. The algorithm can be found in [**22**].

In order to simplify the notation, we will make use of the following definition. Given a bipartite graph G with perfect matchings, we define the set of all its perfect matchings as:

$$\mathcal{M}(G) = \{M : M \text{ is a perfect matching of } G\}. \tag{19}$$

An edge that belongs to a perfect matching is called an *allowed edge*. The problem of finding all the allowed edges in a bipartite graph $G = (U \sqcup V, E)$ consists of obtaining the following subset of edges.

$$E_a = \{uv \in M : M \in \mathcal{M}(G)\}. \tag{20}$$

Note that equivalentely,

$$E_a = \bigcup_{M \in \mathcal{M}(G)} M.$$

There are very efficient algorithms to solve this problem. We can find one with the best time complexity in [**22**]. The time complexity of the algorithm is $O(m)$ given one perfect matching as input. Therefore the time complexity of the overall process is dominated by the time complexity for finding the perfect matching. The research goes a bit further. In [**22**] we can also find an algorithm for obtaining all the edges that occur in at least one maximum cardinality matching. This is useful when the bipartite graph has no perfect matchings. In this work we are only interested in bipartite graphs with perfect matchings, therefore we are only interested in results for perfect matchings. The following proposition summarizes one of the main results of [**22**].

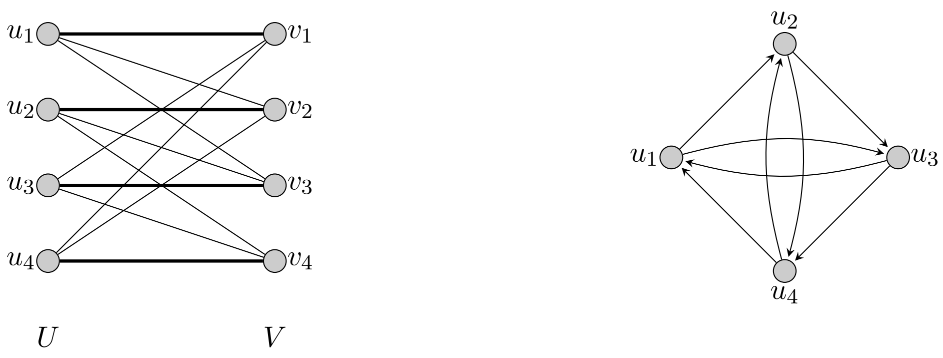
Proposition 2.8 ([**22**]). Given a bipartite graph $G = (U \sqcup V, E)$ and a perfect matching M. All the edges that occur in at least one perfect matching can be found in $O(|E|)$ time.

In the remaining of this section we will focus on summarizing the algorithm of [**22**] for finding the allowed edges in a bipartite graph $G = (U \sqcup V, E)$ with $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$. A cycle is called an *alternating cycle* with respect to a matching M if the cycle alternates between matched and unmatched edges.

Theorem 2.9 ([**22**], *Theorem 2.2*). Let $G = (U \sqcup V, E)$ be a balanced bipartite graph with a perfect matching M, then an edge uv is an allowed edge if and only if it is included in an alternating cycle with M.

Given a perfect matching M, we construct a directed graph induced by $G = (U \sqcup V, E)$ and M. The digraph is given by $D = (U, A)$, with arc set $A = \{(u_i \overleftarrow{M}(v_j)) \,|\, u_i v_j \in E,\ u_i v_j \notin M\}$, where $\overleftarrow{M}(v_j)$ is the mate of v_j under the matching. In summary, D is the digraph resulting from directing all the edges of G from U to V, and then contracting all the arcs that correspond to the matching. See figure 2.1 for an example of this process, where the identity matching is used for simplicity of the example.

In the following corollary we see how to transform the problem of finding edges in alternating cycles to the problem of finding edges in directed cycles in a digraph. It is not hard to prove the veracity of the corollary.



A bipartite graph with a matching in thick edges.

The induced digraph by the matching.

Figure 2.1. Digraph induced by a perfect matching.

Corollary 2.10. Given a bipartite graph $G = (U \sqcup V, E)$ with a matching M, and the respective induced digraph $D = (V', A)$. Then an edge $u_i v_j \in E$ is an allowed edge if and only if the arc $(v_i' v_j') \in A$ is in a directed cycle.

Remember from Proposition 1.3 that an arc of a digraph is in a directed cycle if and only if the arc is in a strongly connected component of the digraph. Therefore our task is reduced to the problem of finding all the strongly connected components of the induced digraph. The problem of finding all the strongly connected components of a digraph $D = (V', A)$ can be solved efficiently by several algorithms, in particular there is an $O(|A|)$ time algorithm due to Tarjan [**24**]. For every arc $(v_i' v_j') \in A$ that is present in a strongly connected component, the respective edge $u_i v_j \in E$ is an allowed edge. We present the complete procedure in the following algorithm, where we use the algorithm of [**24**] for the strongly connected components.

Algorithm 2.11 (Allowed edges in a bipartite graph with perfect matchings, [**22**]).

```
Input: A bipartite graph $G = (U \sqcup V, E)$ and a perfect matching $M$.
Output: All the edges that occur in at least one perfect matching.
Procedure allowed_edges($G, M$)
    $E_a = M$;
    construct the induced digraph $D = (V', A)$;
    find all the strongly connected components of $D$;
    for(all arcs $(v_i' v_j') \in A$ of the strongly connected component) do $E_a = E_a \cup \{u_i v_j\}$;
end
```

Note that every line in the algorithm takes $O(m)$ time. One interesting result is that all the edges of a k-regular bipartite graph are allowed edges. This result follows directly from the Proposition 1.4. An example of this fact is the Figure 2.1.

3. Enumerating all the perfect matchings in a bipartite graph

The problem of enumerating all the perfect matchings of a bipartite graph consists of displaying through some device a list of all the different perfect matchings of a bipartite graph. We do not want to get into much detail so we will only present a brief summary of one of the main results of reference [**20**]. In this reference Takeaki gives an efficient algorithm to enumerate all the perfect matchings of a bipartite graph. His approach to solve this problem is based on a technique developed by himself that he named *trimming and balancing*, which was developed with the intention to be a general technique for speeding up enumeration algorithms. The main result is given in the following theorem.

Theorem 2.12 ([**20**], *Theorem 1*). Perfect matchings in a bipartite graph $G = (U \sqcup V, E)$ can be enumerated in $O(|E|\sqrt{|U|})$ preprocessing time and $O(\log |U|)$ time per perfect matching.

There are older algorithms for solving this problem like the algorithm of Fukuda [**21**]. The algorithm of Fukuda consists of a set of recursion calls. Each recursion call receives as input a balanced bipartite graph $G = (U \sqcup V, E)$ and a perfect matching M of G, and finds a perfect matching different from M. Since we already have a perfect matching, finding a different perfect matching takes $O(|E|)$ time because we only need to find an alternating cycle with respect to M. If no other perfect matching M' is found then we display M. Otherwise we consider an edge uv in $M \backslash M'$ and make two recursion calls in the instances $(G\backslash E_1, M')$ and $(G\backslash E_2, M)$, where $E_1 = \{uv\}$ and $E_2 = \{uv' \mid v \neq v' \in N(u)\}$. The algorithm displays all the perfect matchings since $\mathcal{M}(G) = \mathcal{M}(G\backslash E_1) \cup \mathcal{M}(G\backslash E_2)$. Furthermore, if the algorithm is run in an instance $G = (U \sqcup V, E)$ with $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_n\}$, then each recursion takes $O(m + n)$ time, and finding the initial perfect matching takes $O(m\sqrt{n})$ using Hopcropft-Karp [**13**]. This gives a total time $O(m\sqrt{n} + |\mathcal{M}(G)|(m + n))$.

The algorithm of Takeaki is based on the algorithm of Fukuda combined with the trimming and balancing speed up technique. The difference is that in the trimming technique the graph input at each recursive point is reduced as much as possible. The reduction phase consists of removing all the edges that are not in any perfect matching as in the previous section, as well as removing all the edges that belong to all the perfect matchings of the input. The balancing phase is like a divide and conquer approach, because it consists of balancing the workload between the two subproblems generated at each recursion point. See the reference [**20**] for a detailed version of the algorithm. This approach achieves an amortized time $O(\log(n))$ for each recursion plus the time $O(m\sqrt{n})$ needed to find the initial perfect matching.

CHAPTER 3

The assignment problem

Consider the following optimization problem. We have n persons and n jobs. Each person u is competent to perform a set of jobs $N(u)$. For each person u that can perform a job v there is a weight $w(uv)$, which can be interpreted as the cost incurred for assigning person u to perform job v. The objective is to one-to-one assign all the persons to all the jobs minimizing the total assignment weight. Informally, this is the *Assignment problem*.

More formally, in terms of graph theory, the assignment problem can be stated as follows. We have a bipartite graph $G = (U \sqcup V, E)$, with $n = |U| = |V|$, and an *integer weight function* over the edges of G given by $w : E \to \mathbb{Z}$. The objective is to find a perfect matching of G of minimum weight. We call the pair $\{G, w\}$ an *instance of the assignment problem*, or equivalently an *integer weighted bipartite graph*. A perfect matching of minimum weight is called an *optimum matching*.

In this chapter we will reserve the following symbols. $m = |E|$, $n = |U| = |V|$, and $W = \max_{uv \in E} |w(uv)|$. If we want to use this symbols for another temporal purposes we will let it clear for the reader.

In terms of linear optimization, we can write the assignment problem in a more useful way as a linear program. In this case the decision variables are $\{x_{uv} : uv \in E\}$. And the cost vector is given by the elements of $\{w(uv) : uv \in E\}$. Indeed, $x_{uv} = 1$ if edge uv is in the resulting perfect matching, and is zero otherwise.

The linear program that models the assignment problem, which we call the *assignment program*, is as follows.

$$\text{(21)} \qquad \text{minimize:} \quad \sum_{uv\in E} w(uv)x_{uv}$$

$$\text{(22)} \qquad \text{subject to:} \quad \sum_{v\in N(u)} x_{uv} = 1, \qquad \forall u \in U,$$

$$\text{(23)} \qquad \sum_{u\in N(v)} x_{uv} = 1, \qquad \forall v \in V,$$

$$\text{(24)} \qquad x_{uv} \in \{0,1\}, \qquad uv \in E.$$

Observe that the constraint (22) forces the incident edges of each $u \in U$ to be exactly one, and the constraint (23) does the same thing for each vertex $v \in V$. This two constraints force the feasible solutions to be perfect matchings. Since $x_{uv} = 1$ if and only if the edge uv is in the matching M, then we will abuse the terminology to say under this relation that a perfect matching is a feasible solution of the assignment program and that an optimum matching is an optimal solutions of the assignment program.

Note that the constraint (24) requires the decision variables to be integer. This could be a problem because this type of problems belong to the class of *integer programming problems*, which are a lot more dificult to solve than regular linear programming problems. However, Theorem 1.7 and the following theorem found in [**1**] show that this is not a problem indeed. Because it shows that in this case the constraint (24) is equivalent to the relaxed constraint $x_{uv} \geq 0$, for $uv \in E$. Therefore we end up with a regular linear program in standard form.

Theorem 3.1 ([**1**], *Theorem 2.18*). The vertices of the assignment polyhedron uniquely correspond to integer solutions of the assignment program.

This property can also be derived considering a special case of matrices. An $n \times n$ matrix A is *unimodular* if $det(A) = \pm 1$. And an $n \times m$ matrix A is *totally unimodular* if every regular $k \times k$ submatrix B of A is unimodular. Therefore, if the right-hand side vector b of the system $Bx = b$ is integer valued, then from Cramer's rule follows that the solution x of the system is also integer valued.

Proposition 3.2 ([**1**], *Proposition 2.24*). The coefficients matrix of the assignment program is totally unimodular.

It follows from this proposition and from Theorem 1.8 that every feasible solution of the assignment program is integer valued. Due to this property, the assignment problem can be solved using general linear programming techniques. These techniques are guaranteed to finish in a finite number of steps as Corollary 1.9 implies. But such general techniques perform pretty bad in practice. This has given a wide field of study in the area of optimization algorithms for the development of specialized algorithms to solve the assignment problem exploiting its particular structure. Examples of such specialized algorithms are the Hungarian method [**7**], the Goldberg & Kennedy algorithm [**12**], the Gabow & tarjan algorithm [**11**], the Dinic & Kronrod algorithm [**9**], and the Auction algorithm [**10**]. Some of them are better in theory and some of them are better in practice. In particular, we are interested in the Auction algorithm because it is based on very intuitive concepts and it has proven to be of great practical performance in our experiments in comparison to other algorithms of equal and even better theoretical time complexity.

1. The auction algorithm

The ϵ-scaling Auction Algorithm [**10**] is a method that operates like a real auction. The original algorithm is described in terms of persons and objects, but in this case persons compete for jobs. In the dual program induced by the assignment program, part of the dual variables can be seen as prices attached to the jobs. The optimization process is done in a competitive bidding, where the prices of the jobs are properly modified in order to make the desired job of a person less desirable to the other persons.

The theoretical time complexity of this algorithm is not the best currently known, but we have an implementation that performs a lot better in practice over other algorithms with equal or better theoretical time complexity. The time complexity of the ϵ-scaling auction algorithm is $O(nm\log(nW))$.

In the ϵ-scaling auction algorithm every job v has a price $p(v)$. This set of prices can be seen as a price funciton over V defined by $p : V \to \mathbb{R}$. For every edge $uv \in E$ we define the *reduced cost* of job v for the person u as $w(uv) - p(v)$. The objective in the auction algorithm is to find prices $p(v)$ for the jobs and a perfect matching M such that every person is assigned to the job that gives him almost minimum reduced cost. This condition is expressed in the following definition.

Definition 3.3. Given $\epsilon > 0$. A set of prices p and a perfect matching M are said to satisfy the *ϵ-Complementary Slackness Condition*, or *ϵ-CS condition* for short, if they satisfy that

$$w(uv) - p(v) \leq \min_{v' \in N(u)} \{w(uv') - p(v')\} + \epsilon, \qquad \forall uv \in M.$$

The following theorem shows why this condition is important.

Theorem 3.4. Let $\epsilon > 0$. If a perfect matching M satisfies the ϵ-CS condition with a set of prices p, then M is within $n\epsilon$ of being of minimum weight. That is, if a perfect matching M^* is of minimum weight, then $w(M) \leq w(M^*) + n\epsilon$.

Proof. From the definition of ϵ-CS condition follows that

$$w(M) = \sum_{uv\in M} w(uv) = \sum_{uv\in M} w(uv) - \sum_{v\in V} p(v) + \sum_{v\in V} p(v) = \sum_{uv\in M} (w(uv) - p(v)) + \sum_{v\in V} p(v)$$

$$\leq \sum_{uv\in M^*} (w(uv) - p(v) + \epsilon) + \sum_{v\in V} p(v) = \sum_{uv\in M^*} w(uv) + \sum_{uv\in M^*} \epsilon = w(M^*) + n\epsilon.$$

□

Since we are working on integer weighted bipartite graphs, the following corollary shows how to obtain an optimum matching via the ϵ-CS condition.

Corollary 3.5. If a perfect matching M and a set of prices p satisfy the ϵ-CS condition for $\epsilon < \frac{1}{n}$, then M is of minimum weight.

Proof. If w^* is the optimum weight, then theorem 3.4 implies that $w(M) < w^* + 1$. Since the weights are all integral then $w(M) = w^*$. □

The following pseudo-code shows how to obtain a perfect matching and a set of prices that satisfy the ϵ-CS condition for a given $\epsilon > 0$. If the given instance has perfect matchings, then the procedure always terminates with the correct output as we will see later. It is important to remark that if the instance has no perfect matchings then the procedure will fall into an infinite loop. This algorithm is called the *auction algorithm*.

Algorithm 3.6 (The Auction Algorithm)**.**

```
Input: Bipartite graph $G$, weights $w$, initial prices $p$ and $\epsilon > 0$.
Output: Perfect matching $M$ and prices $p$ that satisfy $\epsilon$-CS.

Procedure get_$\epsilon$CS($G, w, p, \epsilon$)
    $M = \phi$;
    while(there is unassigned persons) do
        take an unassigned person $u$;
        find edges $uv$, $uv'$ with minimum and second minimum reduced cost, respectively;
        $\gamma = (w(uv') - p(v')) - (w(uv) - p(v)) \geq 0$;
        [note: if $u$ has only $1$ neighbor, define $\gamma = \infty$]
        if($v$ is assigned to a person $u'$) remove $u'v$ from $M$;
        append $uv$ to $M$;
        $p(v) = p(v) - \gamma - \epsilon$;
    end;
    return $\{M, p\}$;
end
```

Note that we allow the procedure to receive initial prices p. Such initial prices have no particular restrictions, they can be any real values. The algorithm will automatically adjust them after each iteration and will end up with the correct output as the following proposition shows. However, as we will see later, the initial prices have a big impact on the running time of the procedure.

Proposition 3.7. If the procedure get_ϵCS is applied to a feasible instance of the assignment problem, then the procedure will terminate after a finite number of iterations and will return a matching and a set of prices that satisfy the ϵ-CS condition. Regardless of the initial prices.

Proof. The proof is based on the following loop invariant, which can be easily proven by induction. At the start of each loop iteration, the edges of the matching M satisfy the ϵ-CS condition with the current prices, and at the end of each loop iteration the edges of the new matching M still satisfy the ϵ-CS condition with the new prices.

This invariant is true for the empty matching, and its true in general because the only change we do in each iteration is to the price of one job and at the time the change is done such job is unassigned. Also, the price is decreased in such a way that the new mate of the job satisfies the ϵ-CS condition. Finally, since the price is decreased, then it means that this job will increase its reduced cost for all other persons incident to it, therefore this change will not violate the ϵ-CS condition for the other edges of the matching. This proves the veracity of the loop invariant.

Another observation is that at each iteration, only two things can happen. An unassigned job gets assigned, increasing the size of the matching by one, or an assigned job gets assigned to a different person, letting the size of the matching unchanged. Since after every iteration the price of an assigned job is strictly decreased, then at some point after sufficient decrements to the prices of assigned jobs, one unassigned job will become the best option of one unassigned person. At this point the size of the matching will be increased by one. Therefore after a finite number of steps the matching reaches the maximum cardinality, becoming a perfect matching that satisfies the ϵ-CS condition with the resulting prices. □

Algorithm 3.8 (The ϵ-scaling Auction Algorithm)**.**

```
Input: Bipartite graph $G$ and weights $w$.
Output: Perfect matching $M$ of minimum weight.

Procedure $\epsilon$Scaling($G, w$)
    $p(v) = 0$, for all $v \in V$;
    $\epsilon = W$ and define some $\alpha > 1$;
    do
        $\epsilon = \epsilon/\alpha$;
        $\{M, p\}$ =get_$\epsilon$CS($G, w, p, \epsilon$);
    while($\epsilon \geq 1/n$);
    return $M$;
end
```

There are some relevant notes about the get_ϵCS procedure. Our objective is to find a perfect matching M that satisfies the ϵ-CS condition for $\epsilon < \frac{1}{n}$, because according to the Corollary 3.5 the resulting matching is optimum. It turns out that if the initial prices p are random and ϵ is very close to 0, then the resulting matching will be optimum or very close to optimum but the running time will be huge, and if ϵ is large then the running time will be small but the matching will be far from being optimum. However, if the initial prices have a structure close to ϵ-CS condition prices structure, then the running time is proven to be $O(nm)$ [**10**]. The ϵ-scaling auction algorithm exploits this behavior to efficiently find a perfect matching M that satisfies the ϵ-CS condition for our small $\epsilon < \frac{1}{n}$.

The idea of the ϵ-scaling auction algorithm is to fix a scaling factor $\alpha > 1$. Then iteratively find a sequence of pairs $\{M, p\}$ that satisfy the ϵ-CS condition, where the sequence of values for ϵ is $\{W/\alpha, W/\alpha^2, W/\alpha^3, \ldots, W/\alpha^k\}$, with $W/\alpha^k < 1/n$. As an observation for a better understanding of the algorithm, note that the resulting prices that satisfy the ϵ-CS condition for $\epsilon = W/\alpha^i$, almost satisfy the ϵ-CS condition for the next iteration with $\epsilon = W/\alpha^{(i+1)}$. The procedure is given in the following algorithm. This algorithm is called the *ϵ-scaling auction algorithm*.

As we mentioned before, each call to get_ϵCS takes time $O(nm)$ time. And the number of scaling phases is $O(\log(nW))$. This gives us a total of $O(nm\log(nW))$ running time for the ϵ-scaling auction algorithm. The following corollary follows directly from Proposition 3.7 and from Corollary 3.5.

Corollary 3.9. If the procedure ϵScaling is applied to a feasible instance of the assignment problem, then the procedure will terminate after a finite number of steps and will return an optimum matching.

As we have left clear, this algorithm only solves the assignment problem for perfect matchings in balanced graphs. If we feed any of these algorithms with an unfeasible instance then the algorithms will loop forever. In the Appendix A we can find some efficient techniques to attack the assignment problem when the instance has no perfect matchings and we are interested in maximum cardinality matchings.

CHAPTER 4

Classification of the optimum solutions of the assignment problem

In this chapter we present part of our original contributions. We keep working on feasible balanced bipartite graphs $G = (U \sqcup V, E)$, also we reserve the symbols $m = |E|$, $n = |U| = |V|$, and $W = \max_{uv \in E} |w(uv)|$. If we want to use this symbols for another temporal proposes we will mention it. Recall that we call an optimum matching to a minimum weight perfect matching.

The objective of this chapter is to give efficient algorithms to solve the following three novel problems on weighted bipartite graphs. The problem of finding all the edges that occur in at least one minimum weight perfect matching. The problem of enumerating all the minimum weight perfect matchings. And the *assignment problem with preferences*, which is a variation of the assignment problem. In this problem we provide a weighted bipartite graph and a subset of preference edges $E_{pre} \subseteq E$, and we want to find a minimum weight perfect matching M that maximizes $|M \cap E_{pre}|$. In other words, we want to achieve two different optimizations in one single step. We want to get a minimum weight perfect matching that contains the maximum number of our preference edges.

The assignment problem with preferences has very important applications. Since the minimum weight perfect matching is not necessarily unique, this problem allows us to chose the one that is more useful for us. One possible application is in the following problem. Suppose we have n machines and n tasks with an incurring time $w(uv)$ for machine u to execute task v. We also consider in the problem that some machines can overheat while executing certain tasks. The objective is to find a one-to-one assignment of the machines to the tasks so that the total serial time is minimum and we also want the minimum number of machines to be assigned to overheating tasks, without compromising the minimum time.

Recall that unweighted versions of the first two problems that we address in this chapter have been studied in [**22, 23, 25**].

In advance, assuming that we already have an ϵ-optimal solution of the dual assignment program we can solve the problem of finding all the edges that occur in at least one optimum matching in $O(m)$ time. The problem of enumerating all the optimum matchings can be solved in $O(m\sqrt{n})$ preprocessing time and $O(\log(n))$ time per optimum matching. And the assignment problem with preferences can be solved in $O(m\sqrt{n}\log(n))$ time. In latter sections we will give a formulation of the dual assignment program, and definitions for its ϵ-optimal solutions.

One important observation is that our previous summary was based on the fact that we start with ϵ-optimal solutions of the dual assignment program. We decided to do things this way because in general, all the algorithms that solve the assignment problem provide an optimal solution for the primal assignment program, and also end up with an optimal or ϵ-optimal solution of the dual assignment program, in some cases up to a constant [**11**]. Therefore, if the time complexity for solving the assignment problem improves, then the total time of our algorithms get automatically improved.

1. Optimality conditions for the assignment program

Remember from chapter 3 that the assignment problem can be modeled by the following linear assignment program.

$$\tag{25}\begin{aligned} \text{minimize:} \quad & \sum_{uv\in E} w(uv)x_{uv} \\ \text{subject to:} \quad & \sum_{v\in N(u)} x_{uv} = 1, \qquad \forall u \in U, \\ & \sum_{u\in N(v)} x_{uv} = 1, \qquad \forall v \in V, \\ & x_{uv} \geq 0, \qquad uv \in E. \end{aligned}$$

Also remember that since $x_{uv} = 1$ if and only if $uv \in M$, we will abuse the terminology to say that a perfect matching M is a feasible solution of the assignment program and that an optimum matching is an optimum solution of the assignment program.

It follows from the formulations in (9) that the dual program associated to the assignment program, called the *dual assignment program* is given by the following model, where the pair $P = (\pi, p)$ conform the set of *dual prices*. In particular π assigns prices to the vertices U and p assigns prices to the vertices V. This interpretation of the dual prices is possible because there is one constraint for each $u \in U$ and one constraint for each $v \in V$.

The dual assignment program is given by the model:

$$\tag{26} \text{maximize:} \quad \sum_{u\in U} \pi(u) + \sum_{v\in V} p(v)$$

$$\tag{27} \text{subject to:} \quad \pi(u) + p(v) \leq w(uv), \qquad uv \in E.$$

In what follows we give definitions of optimality and ϵ-optimality conditions for the assignment problem in terms of linear optimization. Derived from the Complementary slackness theorem 1.14, we obtain the following result, which can be found in almost any paper that addresses the assignment problem.

Proposition 4.1. Let M be a perfect matching and $P = (\pi, p)$ be prices of the dual assignment program. Then M is an optimum matching and P are optimal prices of the primal and the dual assignment programs respectively, if

$$\pi(u) + p(v) \leq w(uv), \qquad \forall\, uv \in E, \tag{28}$$

$$\pi(u) + p(v) = w(uv), \qquad \forall\, uv \in M. \tag{29}$$

Proof. In our case, since the assignment program is in standard form and M is a feasible matching, the complementary slackness theorem translates to M and P are optimal if

$$\begin{aligned} \pi(u) + p(v) \leq w(uv), &\qquad \forall uv \in E, \qquad \text{(dual feasibility)} \\ (w(uv) - \pi(u) - p(v))x_{uv} = 0, &\qquad \forall uv \in E, \qquad \text{(slackness)} \end{aligned}$$

Note that in the last condition, $x_{uv} = 1$ if and only if $uv \in M$, therefore it is equivalent to

$$\pi(u) + p(v) = w(uv), \qquad \forall uv \in M.$$

□

Note that optimality is not a definition, it is indeed a property. Following we present the definition of ϵ-optimal solutions, which basically consists in allowing the solutions to violate the optimality condition by a desired amount $\epsilon > 0$.

Definition 4.2. Let $\epsilon > 0$, M be a perfect matching, and $P = (\pi, p)$ be dual prices. We say that M and P are *ϵ-optimal solutions* of the primal and the dual assignment programs respectively, if

$$\pi(u) + p(v) \leq w(uv) + \epsilon, \qquad \forall\, uv \in E, \tag{30}$$

$$\pi(u) + p(v) = w(uv), \qquad \forall\, uv \in M. \tag{31}$$

Note that the definition (4.2) of ϵ-optimality for the primal and the dual assignment programs is equivalent to the Definition 3.3 of the ϵ-complementary slackness condition of the auction algorithm. The proof of this fact can be summarized as follows. Condition (31) lead us to $\pi(u) = w(uv) - p(v)$, for each $uv \in M$. Condition (30) implies that $w(uv) - p(v) \leq w(uv') - p(v') + \epsilon$ for all $v' \in N(u)$. Both combined imply that $\forall uv \in M$, $w(uv) - p(v) = \min_{v' \in N(u)}\{w(uv') - p(v')\} + \epsilon$. The other direction is similar. The following proposition is an equivalent version of Theorem 3.4.

Proposition 4.3. Let M be a perfect matching and $P = (\pi, p)$ be dual prices that are ϵ-optimal. Then M is within $n\epsilon$ of being optimum. That is, if w^* is the optimum weight then $w(M) \leq w^* + n\epsilon$.

Proof. If M^* is a perfect matching of optimum weight w^*. Then it follows directly from the definition of ϵ-optimal solutions that

$$w(M) = \sum_{uv \in M} w(uv) = \sum_{uv \in M} (\pi(u) + p(v)) \leq \sum_{uv \in M^*} (w(uv) + \epsilon) = w(M^*) + n\epsilon.$$

□

If we are interested in primal optimal solutions then we can make use of the following corollary.

Corollary 4.4. If M and P are ϵ-optimal solutions for $\epsilon < 1/n$, then M is optimum.

The main result of this chapter is based on the following observation made by the author, which can be a corollary of the Complementary slackness theorem 1.14. This observation is very important because it is the basis for the characterization of all the optimal solutions of not only the assignment program but any linear program in standard form.

Corollary 4.5. Given a linear program in standard form for minimizing $c \cdot x$ subject to $\{Ax = b, x \geq 0\}$. If p is an optimal dual solution, then a feasible primal solution x is optimal if and only if $x_j = 0$ for all j such that $(c_j - p \cdot A_j) \neq 0$.

Proof. The proof follows directly from the complementary slackness theorem. Observe that in standar form, the complementary slackness theorem translates to: feasible x and p are optimal if and only if $(c_j - p \cdot A_j)x_j = 0$, for all j. Since p is already optimum then x is optimum if and only if $x_j = 0$ whenever $c_j - p \cdot A_j \neq 0$. □

This result is translated to the case of the assignment program in the following corollary.

Corollary 4.6. Given optimum dual prices $P = (\pi, p)$ for the dual assignment program, a perfect matching M is optimum if and only if $w(uv) - \pi(u) - p(v) = 0$ for all $uv \in M$.

Proof. Follows from the fact that in a feasible solution, $x_{uv} = 0$ if and only if $uv \notin M$. □

Observe that Corollary 4.6 is only useful in the presence of optimal prices. However, we do not need to worry about ϵ-optimal solutions, because latter in this chapter we will present an $O(n)$ time algorithm to transform ϵ-optimal solutions with $\epsilon \leq 1/(n+1)$ into optimal solutions.

2. The subgraph G_{cs}

In this section we introduce the most important object of this chapter, the subgraph G_{cs}. This subgraph is important because it reduces problems in weighted perfect matchings to problems in unweighted perfect matchings. This subgraph is given in the following definition.

Definition 4.7. Given optimal dual prices $P = (\pi, p)$, we define the subgraph $G_{cs}(P) = (U \sqcup V, E_{cs})$, which is defined on the same vertex set, but the edge set is given by:

$$E_{cs}(P) = \{uv \in E \mid \pi(u) + p(v) = w(uv)\}.$$

Note that $G_{cs}(P)$ is obtained by removing all the edges uv of G such that $w(uv) - \pi(u) - p(v) \neq 0$. Also note that the subgraph only conserves edges but no weights. The following proposition follows directly from the definition of the G_{cs} subgraph.

Proposition 4.8. The subgraph $G_{cs}(P)$ can be obtained in linear $O(m)$ time.

In order to make notation and proofs more clear, we want to define the following sets.

Definition 4.9. Let us define $\mathcal{M}(G, w)$ as the set of all the optimum perfect matchings of the instance $\{G, w\}$, and define $\mathcal{M}(G_{cs}(P))$ as the set of all the perfect matching of the subgraph $G_{cs}(P)$.

The following theorem states a strong relation between the two sets $\mathcal{M}(G, w)$ and $\mathcal{M}(G_{cs}(P))$. This theorem is indeed the pillar of our algorithms.

Theorem 4.10. If $P = (\pi, p)$ are optimal dual prices, then $\mathcal{M}(G, w) = \mathcal{M}(G_{cs}(P))$.

Proof. From Corollary 4.6 we know that a perfect matching M is optimum if and only if $w(uv) - \pi(u) - p(v) = 0$ for all $uv \in M$, which happens if and only if $M \in \mathcal{M}(G_{cs}(P))$. □

Note that Theorem 4.10 implies that the set $\mathcal{M}(G_{cs}(P))$ is the same for all optimal prices P. However, the following example shows that the subgraph $G_{cs}(P)$ can be different for different optimal prices P.

Example 4.11. Let G be a bipartite graph with $U = \{u_0, u_1, u_2\}$ and $V = \{v_0, v_1, v_2\}$ as in figure 4.1. If $P_1 = (\pi_1, p_1)$, are the optimal prices given by the maps $\pi_1 : \{u_0 \mapsto -2,\, u_1 \mapsto 0,\, u_2 \mapsto 1\}$ and $p_1 : \{v_0 \mapsto 3,\, v_1 \mapsto 1,\, v_2 \mapsto 0\}$ as in figure 4.1(b), then it is not difficult to check that

$$E_{cs}(P_1) = \{u_0v_0,\, u_1v_1,\, u_2v_1,\, u_2v_2\}.$$

Besides, if $P_2 = (\pi_2, p_2)$ are different optimal prices given as in figure 4.1(c), then is not difficult to check that

$$E_{cs}(P_2) = \{u_0v_0,\, u_0v_1,\, u_1v_1,\, u_2v_1,\, u_2v_2\},$$

and as we can see $G_{cs}(P_1) \neq G_{cs}(P_2)$.

It is important to remark that corollary 4.5 can be applied to any linear program in standard form to classify its optimal primal solutions using one optimal dual solution. For instance, if we consider the transportation problem on a bipartite graph G with given supplies, demands, capacities and per-uni costs. Then using one optimal dual solution, we can construct a subgraph G_{cs} of G such that the

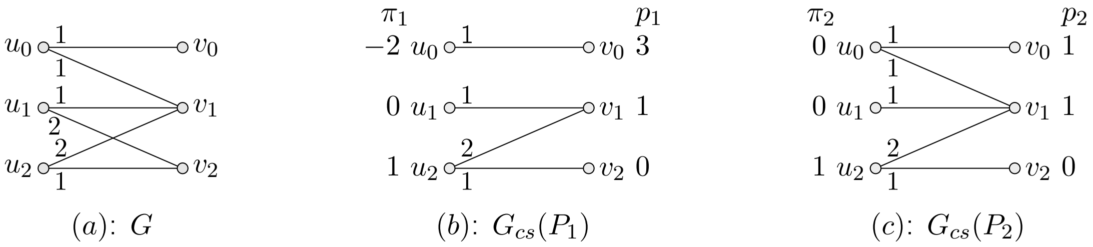


Figure 4.1. **(a)** An integer weighted bipartite graph $\{G, w\}$. **(b)** Optimal prices P_1 and the subgraph $G_{cs}(P_1)$. **(c)** Optimal prices P_2 and the subgraph $G_{cs}(P_2)$.

set of all the optimal flows of the original instance is equal to the set of all the feasible flows on the subgraph G_{cs} under the same supplies, demands and capacities. In the case of the transportation problem there are also algorithms to find optimal dual solutions, but in general it is not always easy to efficiently find optimal dual solutions of a linear program in standard form. However, we can always use linear programing techniques like the simplex method [**3**, Chapter 3] or the dual simplex method [**3**, Chapter 4] to obtain such optimal dual solutions.

2.1. Constructing G_{cs} from ϵ-optimal solutions.

Some of the algorithms that solve the assignment problem, especially those based on cost scaling techniques, solve it by finding ϵ-optimal solutions for a small enough $\epsilon > 0$ that guarantees the optimality of the primal solution, as proposition 4.3 states. See for instance [**10, 11, 12**].

In some cases it is not obvious that an algorithm obtains ϵ-optimal solutions for the primal and the dual linear assignment programs. For instance, the Gabow-Tarjan algorithm [**11**] works as follows. Given a weighted bipartite graph $\{G, w\}$ and $\xi > 0$, they define a new cost function on the edges given by $\hat{w}(uv) = \xi\, w(uv)$ for all $uv \in E$, then for $\epsilon = 1$ they find ϵ-optimal solutions M and $\hat{P} = (\hat{\pi}, \hat{p})$ for the new instance $\{G, \hat{w}\}$. It turns out that if we define the prices $P = (\pi, p)$ as $\pi(v) = \hat{\pi}(u)/\xi$ and $p(v) = \hat{p}(v)/\xi$, then it is easy to prove that M and $P = (\pi, p)$ are $(1/\xi)$-optimal solutions for the original instance $\{G, w\}$. Therefore we can get ϵ-optimal solutions for any $\epsilon > 0$ if we take $\xi = 1/\epsilon$.

As we mentioned before, the subgraph $G_{cs}(P)$ is only useful for optimal prices P. In this section we provide an algorithm to transform ϵ-optimal solutions M and $P_\epsilon = (\pi_\epsilon, p_\epsilon)$ with $\epsilon \leq 1/(n+1)$ into optimal solutions M and $P = (\pi, p)$, in linear time $O(n)$. Observe that the matching M is not changed by the transformation. This is due to the fact that when $\epsilon < 1/n$ the resulting matching is proven to be already optimum as we saw in Corollary 4.4.

In summary, the algorithm finds a value $t \in \{0, \ldots, n\}$ such that if we define

$$p(v) = \lfloor p_\epsilon(v) + t/(n+1) \rfloor, \qquad \forall v \in V, \tag{32}$$

$$\pi(u) = w(uv) - p(v), \qquad \forall uv \in M, \tag{33}$$

then $P = (\pi, p)$ is an optimal dual solution.

As we will prove, it turns out that an appropriate value of t that makes this algorithm work should satisfy that

$$t \neq \lceil (n+1)(\lceil p_\epsilon(v) \rceil - p_\epsilon(v)) \rceil \text{ mod } (n+1), \qquad \forall v \in V. \tag{34}$$

Note that the expression in the right is an integer value in $\{0, \ldots, n\}$. Since $|V| = n$ and there are $n+1$ possibilities for t, then there should be at least one available value for t that satisfies (34).

If $\epsilon \leq 1/(n+1)$, the following algorithm shows how to accomplish our objective.

Algorithm 4.12.

```
Input: ε-optimal solutions $M$ and $P_\epsilon = (\pi_\epsilon, p_\epsilon)$.
Output: Optimal prices $P = (\pi, p)$.

Procedure get_optimal($M, P_\epsilon$)
    good($j$)=true, for all $j = 0, \ldots, n$;
    good($\lceil (n+1)(\lceil p_\epsilon(v) \rceil - p_\epsilon(v)) \rceil$ mod $(n+1)$)=false, for all $v \in V$;
    get $t$ such that good($t$)==true;
    $p(v) = \lfloor p_\epsilon(v) + t/(n+1) \rfloor$ for all $v \in V$;
    $\pi(u) = w(uv) - p(v)$ for all $uv \in M$;
    return $P = (\pi, p)$;
end
```

It is not difficult to see that every line in the procedure get_optimal takes $O(n)$ time. This gives an overall $O(n)$ time. Now its time to prove that the procedure returns indeed optimal prices.

Lemma 4.13. Let $r \in \mathbb{R}$, $n \in \mathbb{Z}$ and $t \in \{0, \ldots, n\}$. If $t \neq \lceil (n+1)(\lceil r \rceil - r) \rceil$ mod $(n+1)$, then $\lfloor r + (t-1)/(n+1) \rfloor = \lfloor r + t/(n+1) \rfloor$.

Proof. Let us define the equipartition of the interval $[r, r+1]$ given by $P_i = r + i/(n+1)$ for $i \in \{0, \ldots, n+1\}$. It is not difficult to see from figure 4.2 that $\lceil r \rceil \in [P_{i-1}, P_i)$ if and only if $i = \lceil (n+1)(\lceil r \rceil - r) \rceil$. Thus, if $t \neq \lceil (n+1)(\lceil r \rceil - r) \rceil$, then $\lfloor P_{t-1} \rfloor = \lfloor P_t \rfloor = \lceil r \rceil - 1$ or $\lfloor P_{t-1} \rfloor = \lfloor P_t \rfloor = \lceil r \rceil$. That is, $\lfloor P_{t-1} \rfloor = \lfloor P_t \rfloor$. We need to consider t with modulus $(n+1)$ because of the symetry of r and $r+1$ when r is integral. □

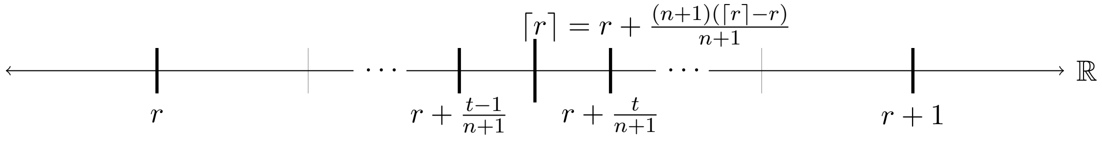


Figure 4.2. The equipartition of the interval $[r, r+1]$ defined by P_i.

Proposition 4.14. Let M and $P_\epsilon = (\pi_\epsilon, p_\epsilon)$ be ϵ-optimal solutions with $\epsilon \leq 1/(n+1)$, then the procedure get_optimal will return optimal prices $P = (\pi, p)$.

Proof. The objective is to prove that the matching and the new prices satisfy a condition equivalent to the optimality condition given in Proposition 4.1. Such equivalent condition is that each $uv \in M$

satisfies that $w(uv) - p(v) \leq w(uz) - p(z)$ for all $z \in N(u)$. The proof of this equivalence is similar to the proof discussed in the comments after the Definition 4.2. Note that the prices π are not present in the condition because this condition is more like the optimality condition of the auction algorithm, nonetheless to complete the dual prices we must define $\pi(u) = w(uv) - p(v)$.

Let $uv \in M$ and $z \in N(u)$. If $z = v$, then the inequality is clear. Thus we can assume that $z \in N(u) \setminus v$. Since M and P_ϵ are ϵ-optimal, then $w(uv) - p_\epsilon(v) \leq w(vz) - p_\epsilon(z) + \epsilon$. Therefore,

$$\begin{aligned}
w(uv)-p(v) &= w(uv)-\lfloor p_\epsilon(v)+t/(n+1)\rfloor \\
&= w(uv)+\lceil -p_\epsilon(w)-t/(n+1)\rceil \\
&= \lceil w(uv)-p_\epsilon(v)-t/(n+1)\rceil \\
&\overset{\epsilon\text{-optimality}}{\leq} \lceil w(uz)-p_\epsilon(z)+\epsilon - t/(n+1)\rceil \\
&\overset{\epsilon\leq 1/(n+1)}{\leq} \lceil w(uz)-p_\epsilon(z)+1/(n+1)-t/(n+1)\rceil \\
&= w(uz)+\lceil -p_\epsilon(z)-(t-1)/(n+1)\rceil \\
&= w(uz)-\lfloor p_\epsilon(z)+(t-1)/(n+1)\rfloor \\
&\overset{\text{(Lemma 4.13)}}{=} w(uz)-\lfloor p_\epsilon(z)+t/(n+1)\rfloor \\
&= w(uz)-p(z).
\end{aligned}$$

Therefore, from proposition 4.1 we get that M and P are optimal solutions of the primal and the dual assignment programs. □

With this algorithm in mind, we can take for granted that we can always get optimal solutions.

Remark 4.15. The current best time complexity to solve the assignment problem, and therefore to find ϵ-optimal solutions for the primal and the dual assignment programs is $O(\sqrt{n}m\log(nW))$. An algorithm that achieves this complexity is given in [**11**].

3. Applications of the subgraph G_{cs}

In this section we will use the subgraph G_{cs} to solve the problems that we described at the beginning of the chapter. The task is now simple because of Theorem 4.10 and because of the algorithms given in [**22, 20, 21**]. Since we have reduced our problems to their respective unweighted variants.

3.1. Finding all the edges that occur in at least one optimum matching

This problem is the weighted variant of the problem addressed in Chapter 2, where we introduced an algorithm for finding all the edges that occur in any bipartite perfect matching. This unweighted version has been addressed by several authors [**22, 23, 25**].

Given a weighted bipartite graph $\{G, w\}$, this problem consists of finding the subgraph of G given by $G_{opt} = (U \sqcup V, E_{opt})$, defined on the same vertex set but with edge set given by:

$$E_{opt} = \{uv \in M \ : \ M \in \mathcal{M}(G, w)\}. \tag{35}$$

Observe that equivalently we can express the subset E_{opt} as follows.

$$E_{opt} = \bigcup_{M \in \mathcal{M}(G,w)} M. \tag{36}$$

Given optimal prices $P = (\pi, p)$, Theorem 4.10 induces a way to compute the subgraph G_{opt}. First, we construct $G_{cs}(P)$ and then we obtain E_{opt} by finding all the edges that occur in at least one perfect matching of $G_{cs}(P)$, which can be done using the algorithm of [**22**] with one of its main results is summarized in Proposition 2.8. Assuming that the instance has perfect matchings, the following algorithm shows how to find G_{opt}.

Algorithm 4.16.

```
Input: An instance ${G, w}$ and optimal dual prices $P$.
Output: The subgraph $G_{opt}$ of $G$.

Procedure get_Gopt($G, w, P$)
    construct the subgraph $G_{cs}(P) = (U \sqcup V, E_{cs})$;
    get $E_{opt} = \{uv \in E_{cs} \mid uv$ is in any perfect matching of $G_{cs}(P)\}$;
    return $G_{opt} = (U \sqcup V, E_{opt})$;
end
```

At this point it should not be dificult to see that the algorithm returns the correct output. However, the following proposition states this more formally.

Proposition 4.17. The procedure *get_Gopt* returns the subgraph G_{opt} from (36) of a weighted bipartite graph.

Proof. It is not dificult to see that the procedure returns indeed $\bigcup_{M \in \mathcal{M}(G_{cs}(P))} M$. But from Theorem 4.10 we know that $\mathcal{M}(G_{cs}(P)) = \mathcal{M}(G, w)$. Therefore the procedure returns $\bigcup_{M \in \mathcal{M}(G,w)} M$. □

Theorem 4.18. Given a weighted bipartite graph $\{G, w\}$ and optimal dual prices P, all the edges of G that occur in at least one minimum weight perfect matching can be obtained in $O(m)$ time.

Proof. Analyzing the steps of the Algorithm 4.16, we get from proposition 4.8 that the subgraph $G_{cs}(P)$ can be obtained in $O(m)$ time, and from proposition 2.8 that E_{opt} can be computed in $O(|E_{cs}(P)|)$ time. This gives an overall $O(m)$ time. □

3.2. Classification of the edges of a weighted bipartite graph

In this section we will briefly describe an interesting application of what we have constructed at this point. We will classify the edges of a weighted bipartite graph in the following disjoint sets.

- *Permanent edges*: This subset of edges is integrated by the edges of G_{opt} that have no adjacent edges or equivalently, edges that are adjacent to vertices of degree one. This type of edges are present in every optimal matching. Formally, this set is:

$$E_{per} = \{uv \in E_{opt} \,:\, deg_{G_{opt}}(u) = deg_{G_{opt}}(v) = 1\}.$$

- *Replaceable edges*: This subset of edges is integrated by the edges of G_{opt} that have at least one adjacent edge. This type of edges are present in at least one optimum matching, and are missing in at least one optimum matching. The set is given formally as:

$$E_{rep} = \{uv \in E_{opt} \,:\, deg_{G_{opt}}(u) = deg_{G_{opt}}(v) > 1\}.$$

- *Forbidden edges*: This subset of edges is integrated by the edges of G that are not in G_{opt}. This type of edges never occur in an optimum matching. Formally, this set is given by:

$$E_{for} = E \backslash E_{opt}.$$

Note that $E_{per} \cup E_{rep} = E_{opt}$, and therefore $E_{per} \cup E_{rep} \cup E_{for} = E$. Another important property of a weighted bipartite graph that can be derived from the construction of G_{opt} is in the following corollary which needs no proof.

Corollary 4.19 (*Uniqueness*). A weighted bipartite graph has a unique optimum matching if and only if G_{opt} has n edges.

It is important to remark that this classification of the edges as well as the uniqueness property of the optimum solutions can be obtained in $O(m)$ time starting from optimal dual prices $P = (\pi, p)$.

3.3. Enumerating all the minimum weight perfect matchings

The problem that we want to solve in this section is the weighted variant of the problem addressed in Chapter 2, where we showed how to enumerate all the bipartite perfect matchings. In this case we want to enumerate all the optimum perfect matchings of a weighted bipartite graph. The direct algorithm derived from Theorem 4.10 is the following. Given optimal dual prices P, the first step is to find the subgraph $G_{cs}(P)$ and then we enumerate all the perfect matchings of $G_{cs}(P)$ which can be done by algorithms like [**20, 21**] as Theorem 2.12 states. In the following algorithm we give the procedure to solve this problem.

Algorithm 4.20.

```
Input: An instance $\{G, w\}$ and optimal dual prices $P$.
Output: The enumeration of the minimum weight perfect matchings of $\{G, w\}$.

Procedure enumerate_MW_Per_Mat($G, w, P$)
    construct the subgraph $G_{cs}(P) = (U \sqcup V, E_{cs})$;
    enumerate all the perfect matchings of $G_{cs}(P)$;
    return;
end
```

Proposition 4.21. The procedure enumerate_MW_Per_Mat enumerates all the minimum weight perfect matchings of a weighted bipartite graph $\{G, w\}$.

Proof. We are enumerating all the elements of $\mathcal{M}(G_{cs}(P))$, but we know from theorem 4.10 that $\mathcal{M}(G_{cs}(P)) = \mathcal{M}(G, w)$. □

Theorem 4.22. Given a weighted bipartite graph $\{G, w\}$ and optimal dual prices P, all its minimum weight perfect matchings can be enumerated in $O(m + |E_{cs}(P)|\sqrt{n} + \mathcal{M} \log n)$ time, where $\mathcal{M}$ is the number of minimum weight perfect matchings.

Proof. Analyzing the steps of the Algorithm 4.20, we get from proposition 4.8 that the subgraph $G_{cs}(P)$ can be obtained in $O(m)$ time, and from theorem 2.12 that we can enumerate all the perfect matchings of $G_{cs}(P)$ in $O(|E_{cs}(P)|\sqrt{n} + \mathcal{M} \log n)$ time, giving an overall $O(m + |E_{cs}(P)|\sqrt{n} + \mathcal{M} \log n)$ time. □

Just as a justification of the long term in the running time. Note that the subgraph $G_{cs}(P)$ can have a lot less edges than G and also note that there can be a very small number of optimum perfect matchings. Therefore, it is possible that every term in $O(m + |E_{cs}(P)|\sqrt{n} + \mathcal{M} \log n)$ dominates the complexity.

3.4. The assignment problem with preferences

The *assignment problem with preferences* is a novel problem that can be stated informally in the following application example. Suppose that we want to one-to-one assign resources to jobs at a minimum total cost as in a traditional assignment problem. We also know that over the jobs that each resource is capable of perform, he has some jobs of his preference that will make him happier if he ends up performing one of this jobs. The objective in this case is to obtain, over all the minimum cost assignments, the one that maximizes the persons assigned to a job of his preference. In other words, maximize the preference as long as the optimality of the assignment cost allows it.

Formally, we have as input a weighted bipartite graph $\{G, w\}$ and a subset of edges E_{pre} of G which can be seen as a set of assignment preferences. The assignment problem with preferences consists of finding an optimum matching M such that $|M \cap E_{pre}|$ is maximum over all the optimum matchings of $\{G, w\}$. In other words, there is no other optimum matching that has more edges of E_{pre} than M. Note that there can be several optimum matchings that respect a maximum number of preferences.

This problem can be easily solved with the help of the subgraph G_{cs}. Given the subgraph $G_{cs}(P)$ for some optimal dual prices P, we define a new weight function $w_p : E(G_{cs}(P)) \to \{0, 1\}$ over the edges of $G_{cs}(P)$, as follows

$$w_p(uv) = \begin{cases} 0 & \text{if } uv \in E_{pre} \\ 1 & \text{if } uv \notin E_{pre}. \end{cases} \tag{37}$$

The algorithm for solving this problem can be derived from the following proposition.

Proposition 4.23. If M is a minimum weight perfect matching of the new instance $\{G_{cs}(P), w_p\}$, then M is an optimum matching of the original instance $\{G, w\}$ such that $|M \cap E_{pre}|$ is maximized over the elements of $\mathcal{M}(G, w)$.

Proof. Since P are optimal dual prices, then from theorem 4.10 follows that M is an optimum matching of $\{G, w\}$. Therefore since $\mathcal{M}(G_{cs}(P)) = \mathcal{M}(G, w)$, then M maximizes $|M \cap E_{pre}|$ over $\mathcal{M}(G, w)$ if and only if there is no other perfect matching of $G_{cs}(P)$ with more edges of E_{pre} than M.

If M' is a perfect matching of $G_{cs}(P)$ containing more edges of E_{pre} than M, then the weight of M' is smaller than the weight of M in the instance $\{G_{cs}(P), w_p\}$, because M' has more edges with zero weight than M. This contradicts the optimality of M on $\{G_{cs}(P), w_p\}$. Therefore M maximizes $|M \cap E_{pre}|$ over $\mathcal{M}(G, w)$. □

In the following algorithm we give the complete procedure to solve this problem.

Algorithm 4.24.

Input: An instance $\{G, w\}$, optimal dual prices P, and a subset of edges E_{pre}.
Output: A minimum weight perfect matching that contains a maximum number of edges of E_{pre}.

```
Procedure AP_preferences($G, w, P, E_{pre}$)
    construct the subgraph $G_{cs}(P) = (U \sqcup V, E_{cs})$;
    construct the wegith function $w_p$, as in (37);
    get a minimum wegith perfect matching $M$ of $\{G_{cs}(P), w_p\}$;
    return $M$;
end
```

The output of the algorithm is the correct as is shown in the following proposition.

Proposition 4.25. The perfect matching returned by the procedure AP_preferences is of minimum weight and maximizes $|M \cap E_{pre}|$ over the elements of $\mathcal{M}(G, w)$.

Proof. Follows directly from proposition 4.23. □

Theorem 4.26. Given a weighted bipartite graph $\{G, w\}$, optimal dual prices P and a subset of edges E_{pre}. A minimum weight perfect matching that maximizes $|M \cap E_{pre}|$ over the set of minimum weight perfect matchings, can be found in $O(m + AP\_01(G_{cs}, w_p))$ time. Where $AP\_01(G_{cs}, w_p)$ is the time needed to solve the assignment problem in an instance with $\{0, 1\}$-weights.

Proof. Analyzing the steps of the Algorithm 4.24, we get from proposition 4.8 that the subgraph $G_{cs}(P)$ can be obtained in $O(m)$ time. And the weight function can be constructed in $O(m)$ time. This gives an overall $O(m + AP\_01(G_{cs}, w_p))$ time. □

Currently the fastest algorithm that solves an instance of the assignment problem with only $\{0,1\}$-weights is the algorithm found in [**11**], which solves a general assignment problem but in this case $W = 1$. This result and Theorem 4.26 imply the following corollary.

Corollary 4.27. Given a weighted bipartite graph $\{G, w\}$, optimal dual prices P and a subset of edges E_{pre}. A minimum weight perfect matching that maximizes $|M \cap E_{pre}|$ over the set of minimum weight perfect matchings, can be found in $O(\sqrt{n}m\log(n))$ time.

An important observation is that the assignment problem with preferences works on an instance of the assignment problem. And the instance $\{G_{cs}(P), w_p\}$ is also an instance of the assignment problem. Therefore we can nest this algorithm to model hierarchy levels in the preferences. We could for example, define preferences sets $E_{pre\_1}, E_{pre\_2}, \ldots, E_{pre\_k}$. In the first phase we consider all the preferences $E_{pre\_1}, \ldots, E_{pre\_k}$ to get the first subinstance $\{G_{cs\_1}, w_{p\_1}\}$. Note that an optimal solution in this first subinstance is a matching that maximizes the preferences from all the sets of preferences. If we now apply the preferences algorithm to the instance $\{G_{cs\_1}, w_{p\_1}\}$ this time only considering the preferences $E_{pre\_2}, \ldots, E_{pre\_k}$. Then we get an second subinstance $\{G_{cs\_2}, w_{p\_2}\}$ which in turn satisfies that an optimum solution of it is such that from all the optimum matchings that maximize the number of preferences from $E_{pre\_1}, \ldots, E_{pre\_k}$ is the one that maximizes the preferences from $E_{pre\_2}, \ldots, E_{pre\_k}$. And we can continue until we process all levels. For the purpose of illustration, lets consider the following example. Let us say that we want to assign persons to tasks and the persons have given their preferences of the tasks they would like to do. We still do not want to compromise the optimality of the solution and we want to maximize the prefrerences. Besides, it turns out that some of the persons are our friends and what we want to do is, from all the optimum solutions that maximize the overall preferences we want to find the one that maximizes the preferences of our friends. This problem can be solved using the approach just described above with two levels of preferences.

All the algorithms and methods described in this chapter only apply to bipartite graphs with perfect matchings. If we want to attack problems in bipartite graphs with no perfect matchings then we can make use of the transformations given in the Appendix A.

4. Performance analysis

In this section we present information about the performance of the algorithm for finding the subgraph G_{opt} described previously in the applications of the subgraph G_{cs}, as the subgraph that has edge set equal to all the edges that occur in at least one optimum matching. The instances were solved in a Windows 8.1 PC with 64 bits architecture, Intel Core i5-4670 CPU @ 3.40GHz and 16GB of RAM.

The idea is not to just show its performance, which is good, but also compare it to when we only solve the regular assignment problem on the same instance. It is important to mention that the underlying algorithm used for solving the assignment problem, and therefore obtaining the ϵ-optimal dual prices, was the Auction algorithm of Bertsekas [**10**] that we presented in Chapter 3.

Since obtaining data of real life scenarios that find application in the assignment problem is difficult, we will test the performance in randomly generated instances.

In the Chapter 5, we will discuss a few models to generate random instances of the assignment problem. Therein we describe a model that we call the Dispersed-degree model. This model is designed to introduce randomness in the graph structure itself rather than in the edge weights, which are distributed uniformly at random in the integer range $\{0, \ldots, 10^9\}$. For each graph generated under this model we use 4 parameters given by (n, s, d, r). The parameters n, s are the number of vertices in the vertex sets U and V respectively, since we will only work on balanced instances then $n = s$ for our tests. The value $d \in [0, 1]$ is a real number such that the resulting graph satisfies $d \approx \dfrac{|E|}{n * s}$, in other words d measures the proportion of edges in the graph respect to the complete bipartite graph with the same vertices. And $r \in [0, 1]$ is a real number that defines how dispersed are the degrees of the vertices of U. For each vertex of U, we define its degree to be an integer random number uniformly distributed in the interval $[d \cdot s - \hat{r}, d \cdot s + \hat{r}]$, where $\hat{r} = r \cdot s \cdot \min(d, 1 - d)$. In other words, $r = 0$ means all vertices of U have the same degree $d \cdot s$, while $r = 1$ means that the degrees of U are as dispersed as possible whithout violating the density.

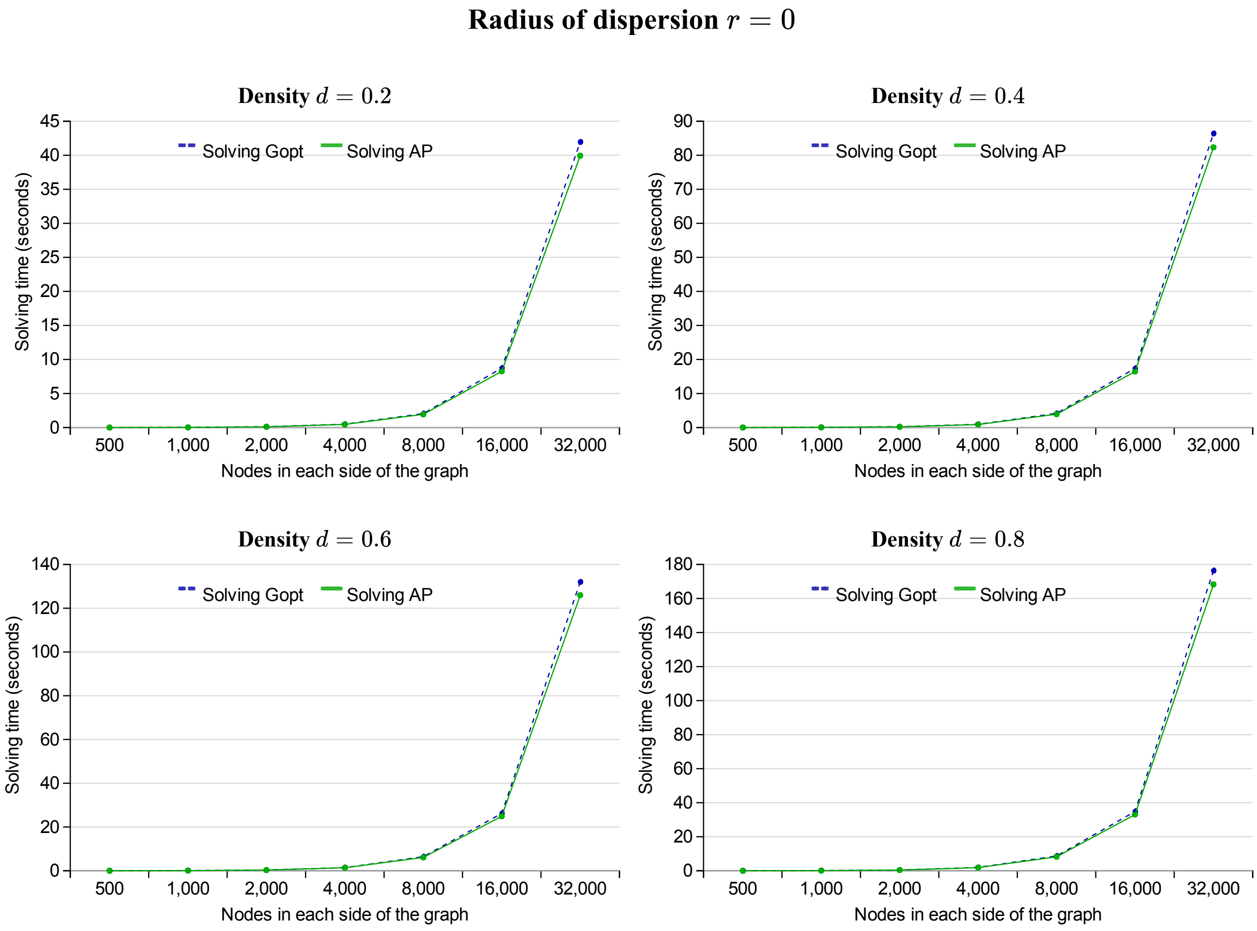


Figure 4.3. Time performance of solving the assignment problem vs solving the problem of finding G_{opt} in the same instance, with radius of dispersion $r = 0.0$, different levels of density and different graph sizes.

We present performance data in plots for different sequences of parameters of the Dispersed-degree model. It is important to remark that, since for each fixed set of parameters we obtain different random instances every time we request for one, then in order to reduce bias in the results we have generated 15 different random instances for each fixed set of parameters. Therefore, what we present is the average solving time of 15 random instances with the same configuration.

In Figure 4.3 we fix the radius of dispersion in $r = 0$, each plot in turn fixes its own value of density, and since we only consider balanced graphs $(n = s)$ the only free parameter is the number of vertices in each side of the graph, this parameter varies along the horizontal axis in every plot. Note that the horizontal axis is in a logarithmic scale. Each plot shows two lines, the solid one for the time taken to solve a regular assignemnt problem from scratch, and the dashed one for the time taken to find the subgraph G_{opt} from scratch in the same instance. As we can observe, the lines are pretty close to each other.

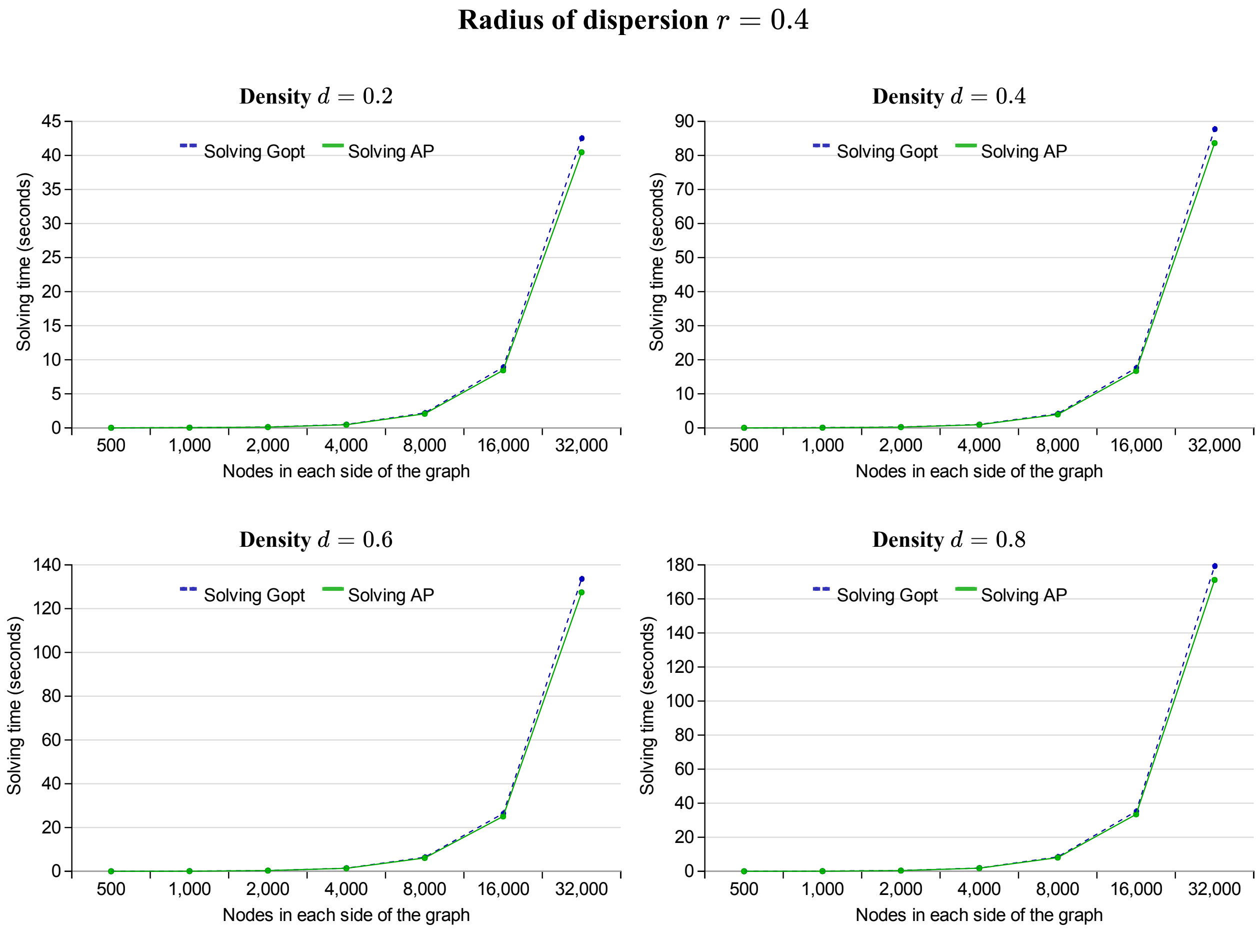


Figure 4.4. Time performance of solving the assignment problem vs solving the problem of finding G_{opt} in the same instance, with radius of dispersion $r = 0.4$, different levels of density and different graph sizes.

Figure 4.4 shows similar information, but this time for the fixed radius of dispersion $r = 0.4$. It might be difficult to see from the figures, but the lines are a bit closer now to each other.

Finally, in Figure 4.5 we show similar information about the time performance, but this time for the fixed radius of dispersion $r = 1.0$, which is the maximum possible dispersion. Although this figure shows the smallest difference between the lines from all the figures, in all the cases the lines show very small difference.

Besides the small time difference between the two problems, it is impressive that we can solve an instance of the assignment problem with $819,200,000$ variables (edges) in about three minutes.

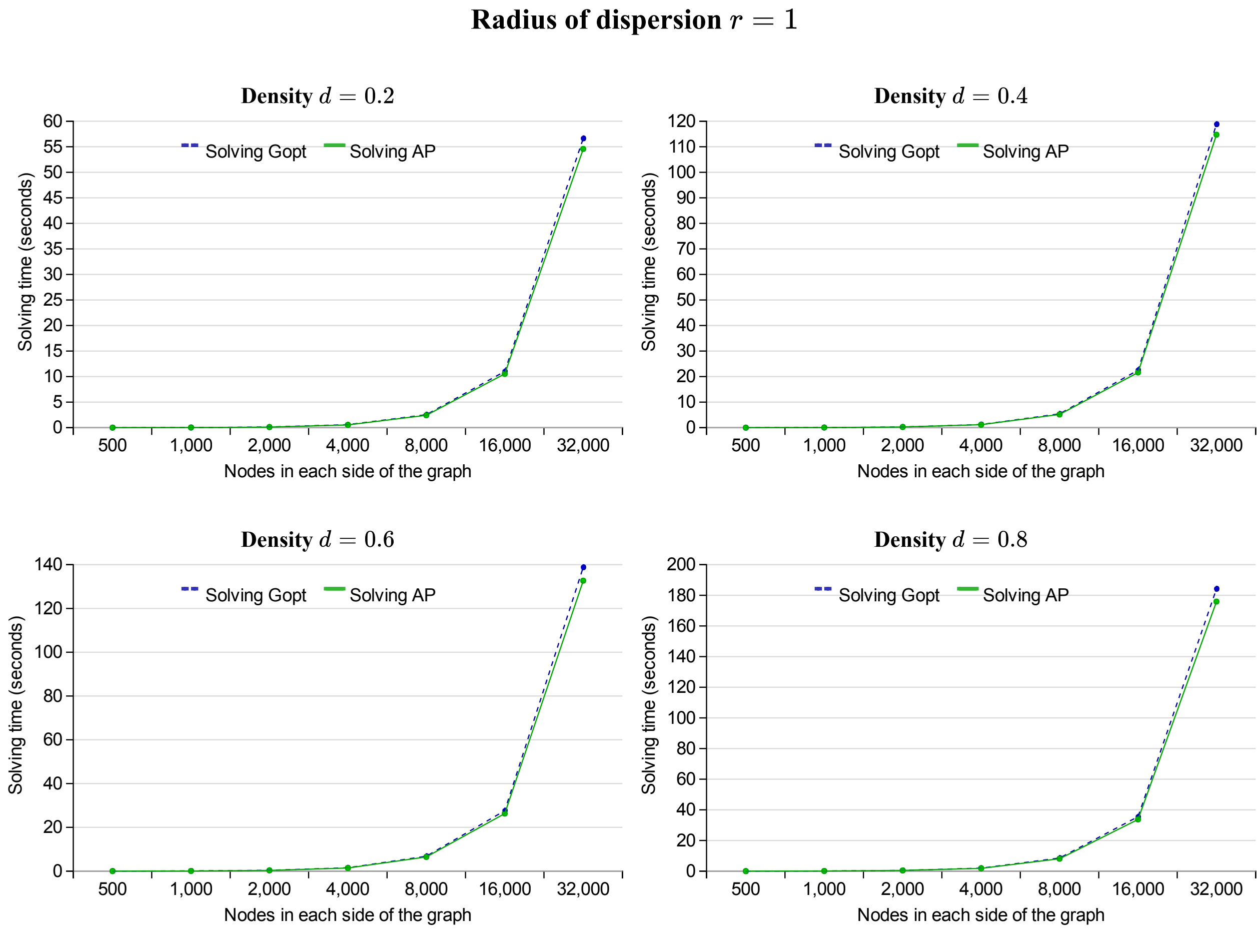


Figure 4.5. Time performance of solving the assignment problem vs solving the problem of finding G_{opt} in the same instance, with radius of dispersion $r = 1.0$, different levels of density and different graph sizes.

Our hypotheses is that since the steps for finding G_{opt} after obtaining the ϵ-optimal solutions are performed in the graph structure, then the more difficult the instance is for the assignment problem solver the smaller the difference in time between the two problems will be. This fact explains the behavior of the lines getting closer to each other when the radius r increases, i.e. making the assignment problem instances more difficult to solve. Even though the problems generated by the Dispersed-degree model are instances with medium level of dificulty for the Auction algorithm, the difference in solving time between the two problems is very small. In conclusion, finding a random optimum matching by means of solving the assignment problem is equally hard than finding all the optimum matchings in both theory and practice. A similar conclusion is expected from the Assignment problem with preferences.

CHAPTER 5

The random assignment problem

The *Random Assignment Problem* consist of an instance $\{G = (U \sqcup V, E), w : \mathbb{R} \to E\}$ of the assignment problem where the weights $w(uv)$ of the edges $uv \in E$ are given independent randomly distributed. Until now, the random assignment problem has been studied only for the complete bipartite graph. In this chapter we study the random assignment problem for a not necessarily complete bipartite graph, more precisely when G is an Erdős-Renyi bipartite graph.

Random assignment problems have been studied exhaustively for many authors in the last years. See [**30**] and the references contained there for an overview of the last results and developments in several types of random assignment problems. The analysis of random assignment problems is very important in order to understand the properties of optimal solutions and feasible regions of assignment problems, specially in large scale cases.

This chapter is organized as follows: we begin by recalling the most important known results about the random assignment problem on the complete bipartite graph $K_{n,s}$, see for instance [**31**] and the references contained there. In Section 2, we introduce the Erdős-Renyi model of a random bipartite graph and some of their variants. After that we pose some conjectures about the random assignment problem, but this time on non-complete bipartite graphs. We finish this chapter by presenting a proof for the first of these conjectures; which establish the expectation value of the minimum weight of a matching of size k for an Erdős-Renyi random bipartite graph. The proof presented is similar to the given in [**31**] for complete bipartite graphs.

1. The random assignment problem for a complete bipartite graph

In this section we present the most important known results about the random assignment problem on the complete bipartite graph $K_{n,s}$. Without loss of generality as shown in [**31**], we assume that the weights of the edges are given independent exponentially distributed with rate 1. We denote the minimum weight of a matching of size k in the complete bipartite graph $K_{n,s}$ as $C_{k,n,s}$. In particular, let $C_n = C_{n,n,n}$. Let $\zeta(s)$ be the Riemann zeta function given by $\sum_{i=1}^{\infty} \frac{1}{i^s}$. As before, we assume that $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_s\}$. Now, we present the most remarkable results for the random assignment problem on the complete bipartite graph.

(38) It was proven that $E[C_n] \to \dfrac{\pi^2}{6} = \zeta(2)$, as $n \to \infty$.

(39) In general it was proven that $E[C_n] = \displaystyle\sum_{i=1}^{n} \frac{1}{i^2}$, which clearly converges to $\zeta(2)$.

(40) Moreover, in **[31]** was proved that $E[C_{k,n,s}] = \displaystyle\sum_{\substack{i,j\geq 0\\ i+j<k}} \frac{1}{(n-i)(s-j)}$.

It is not difficult to prove by induction that $E[C_{n,n,n}]$ reduces to $\displaystyle\sum_{i=1}^{n} \frac{1}{i^2}$, as we prove in the following proposition.

Proposition 5.1. $E[C_{n,n,n}] = \displaystyle\sum_{\substack{i,j\geq 0\\ i+j<n}} \frac{1}{(n-i)(n-j)} = \sum_{i=1}^{n} \frac{1}{i^2} = E[C_n]$.

Proof. The base case of the induction is very easy to verify. Following we show the first three cases for $n = 1, 2, 3$. In each case we have all the valid values for i and for j, and in the blue shaded triangular array we show the terms $\frac{1}{(n-i)(n-j)}$ that will be added up in the summation.

- $n = 1$:

0	1
i j	0

$$E[C_{1,1,1}] = 1.$$

- $n = 2$:

1	1/2	
0	1/4	1/2
i j	0	1

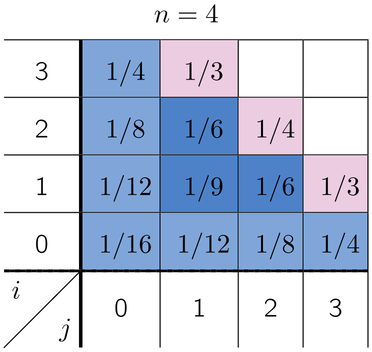
$$E[C_{2,2,2}] = \frac{1}{4} + \frac{1}{2} + \frac{1}{2} = 1 + \frac{1}{2^2}.$$

- $n = 3$:

2	1/3		
1	1/6	1/4	
0	1/9	1/6	1/3
i j	0	1	2

$$E[C_{3,3,3}] = \frac{1}{9} + \frac{1}{4} + \frac{2}{6} + \frac{2}{3} = 1 + \frac{1}{2^2} + \frac{1}{3^2}.$$

The inductive step is based on the observation represented in the following figure. We express the concept in the case $n = 4$, but its not difficult to see that it is true for any n.



$n = 4$

3	1/4	1/3		
2	1/8	1/6	1/4	
1	1/12	1/9	1/6	1/3
0	1/16	1/12	1/8	1/4
i / j	0	1	2	3

Following with the case $n = 4$ in the figure, the idea is to append to the array, the corresponding values of the pink shaded semi-diagonal where $i, j \geq 1$, $i + j = 4$. This makes the array $i, j \geq 1$, $i + j \leq 4$ to be identical to the array of the case $n = 3$, which is composed by the dark-blue and pink shaded cells. Observe that, because of the inductive step for $n = 3$, such entries together with the cell $(i = 0, j = 0)$ complete the summation $\sum_{i=1}^{4} \frac{1}{i^2}$. Therefore, we only need to prove that the pink entries sum equal to the bottom row and left column cells wihtout the cell $(i = 0, j = 0)$, which is very easy to verify it in the figure.

For general n, the sum of the appended pink diagonal can be expressed by $A = \displaystyle\sum_{i=1}^{n-1} \frac{1}{i(n-i)}$. Due to the symmetry of the array, the sum of the bottom row plus the left column, without the cell $(i = 0, j = 0)$, can be expressed by $B = 2\displaystyle\sum_{i=1}^{n-1} \frac{1}{n(n-i)}$. In order to establish formally the identity of the proposition, we prove following that $A = B$.

$$A = \sum_{i=1}^{n-1} \frac{1}{i(n-i)} = \sum_{i=1}^{n-1} \frac{1}{ni} + \sum_{i=1}^{n-1} \frac{1}{n(n-i)} = 2\sum_{i=1}^{n-1} \frac{1}{n(n-i)} = B$$

Assuming that the inductive step is true for $n-1$, i.e. $E[C_{n-1,n-1,n-1}] = \sum_{i=1}^{n-1} \frac{1}{i^2}$, we prove that it is also true for n. To simplify the proof lets denote the summation terms by $t_{n,i,j} = \frac{1}{(n-i)(n-j)}$.

$$\begin{aligned}
E[C_{n,n,n}] &= \sum_{\substack{i,j\geq 0\\ i+j<n}} t_{n,i,j} = \sum_{\substack{i,j\geq 1\\ i+j<n}} t_{n,i,j} + \sum_{\substack{i=0 \text{ or } j=0\\ i+j<n}} t_{n,i,j} \\
&= \sum_{\substack{i,j\geq 1\\ i+j<n}} t_{n,i,j} + \underbrace{2\sum_{i=1}^{n-1} \frac{1}{n(n-i)}}_{B} + \frac{1}{n^2} = \sum_{\substack{i,j\geq 1\\ i+j<n}} t_{n,i,j} + \underbrace{\sum_{i=1}^{n-1} \frac{1}{i(n-i)}}_{A} + \frac{1}{n^2} \\
&= \sum_{\substack{i,j\geq 1\\ i+j<n+1}} t_{n,i,j} + \frac{1}{n^2} = \underbrace{\sum_{\substack{i,j\geq 0\\ i+j<n-1}} t_{n-1,i,j}}_{E[C_{n-1,n-1,n-1}]} + \frac{1}{n^2} = \sum_{i=1}^{n} \frac{1}{i^2} = E[C_n].
\end{aligned}$$

□

2. The non-complete random assignment problem

In this section we briefly describe the necessary definitions to understand the problem that we are addressing. Given a bipartite graph $G = (U \sqcup V, E)$, with $n = |U|$, $s = |V|$ and $m = |E|$, we define its *density* by $\rho(G) = \dfrac{m}{n \cdot s} \in [0, 1]$. Note that in a complete bipartite graph $K_{n,s}$ we have $m = n \cdot s$, and therefore $\rho(K_{n,s}) = 1$. We denote the degree of a vertex x by $deg(x)$. Following we will describe two models to construct bipartite graphs of a given density $d \in [0, 1]$.

2.1. Random bipartite graph models

In the *Erdõs-Renyi* model every possible edge of the bipartite graph has under a bernoulli distribution, probability d to stay in the graph and $(1-d)$ to be taken out of the graph. The pseudocode to generate this type of random bipartite graphs is given in the following procedure. Which takes as argument the number of vertices in U, the number of vertices in V and the desired density d.

```
Procedure erdos_renyi($n, s, d$):
$U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E = \phi$;
for each $u \in U$, $v \in V$:
    if (getBernoulli(d)==1) add edge $uv$ to $E$;
return $G = (U \sqcup V, E)$;
```

The function *getBernoulli(d)* returns 1 with probability d and 0 with probability $(1-d)$. It is not difficult to see that in the resulting graph $\rho(G) \approx d$. Observe that in this model for generating bipartite graphs, the degrees of the vertices satisfy that $deg(u) \approx d \cdot s$ for $u \in U$ and $deg(v) \approx d \cdot n$ for $v \in V$. In other words, we get almost null variability in the distribution of the degrees in each side.

The second model, the *Dispersed-degree* model, is designed to make the distribution of the degrees in the side U more variable while preserving the desired density d. Basically, we define the degree of each vertex $u \in U$ as an integral uniform random number in an interval centered at $d \cdot s$. This interval is of the form $[d \cdot s - r, d \cdot s + r]$, where r is a custom *radius of dispersion*. Note that since $deg(u) \in [0, s]$, then r must satisfy $r \leq s \cdot \min(d, 1-d)$ to keep the degrees in the valid range. Also note that since the interval is centered in $d \cdot s$, then $E[deg(u)] = d \cdot s$, therefore $\rho(G) \approx d$. Once we have defined the degree of a vertex $u \in U$, we take its neighbors as a uniformly random subset of V of size $deg(u)$.

The following pseudocode shows how to generate this class of graphs. It takes as input the number of vertices in U, the number of vertices in V, the desired density d, and the custom radius of dispersion $0 \leq r \leq s \cdot \min(d, 1-d)$.

```
Procedure dispersed_degree($n, s, d, r$):
$U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_n\}$, $E = \phi$;
for each $u \in U$ define $deg(u) := rand(d \cdot s - r, d \cdot s + r)$;
for each $u \in U$:
    take $deg(u)$ uniform random elements of $V$ as neighbors of $u$;
return $G = (U \sqcup V, E)$;
```

The function *rand(a,b)* returns an uniform random integral number in the interval $[a, b]$. The random subset of neighbors of each $u \in U$ can be constructed by shuffling the set V uniformly at random and then take the first $deg(u)$ elements as neighbors of u.

In both models, the weights of the edges are given independent exponentially distributed of rate $\lambda = 1$.

2.2. Conjectures for the non-complete random assignment problem

According to our random models, we denote $C^d_{k,n,s}$ as the minimum weight of a matching of size k in an Erdõs-Renyi(n, s, d) bipartite graph. We also denote $C^{d,r}_{k,n,s}$ as the minimum weight matching of size k in a Dispersed-degree(n, s, d, r) bipartite graph.

The conjectures that we present following are based on experimental results for the case $n = s = k$, that is, the bipartite graph is balanced and the matching is perfect.

Conjecture 5.2. Based on experimental results, it seems that:

- $E[C^d_{k,n,s}] = \dfrac{1}{d} \cdot \displaystyle\sum_{\substack{i,j \geq 0 \\ i+j<k}} \frac{1}{(n-i)(s-j)}$, for an Erdõs-Renyi$(n, s, d)$ graph.
- $E[C^{d,0}_{k,n,s}] = \dfrac{1}{d} \cdot \displaystyle\sum_{\substack{i,j \geq 0 \\ i+j<k}} \frac{1}{(n-i)(s-j)}$, for a Dispersed-Degree$(n, s, d, r = 0)$ graph.

Based on weak experimental results, it seems that:

- $E[C^{d,r}_{k,n,s}] = \dfrac{1}{d} \cdot \displaystyle\sum_{\substack{i,j \geq 0 \\ i+j<k}} \frac{1}{(n-i)(s-j)}$, for a Dispersed-Degree$(n, s, d, r)$ graph.

Note that when $n = s = k$, the expected value reduces to $\dfrac{1}{d} \cdot \displaystyle\sum_{i=1}^{n} \frac{1}{i^2}$, which converges to $\dfrac{\zeta(2)}{d}$.

In this section we will give a proof for the first conjecture, which corresponds to the general Erdõs-Rengy(n, s, d) bipartite graph. That is, we will prove that $E[C^d_{k,n,s}] = \dfrac{1}{d} \cdot \displaystyle\sum_{\substack{i,j \geq 0 \\ i+j<k}} \frac{1}{(n-i)(s-j)}$.

2.3. Proof for the Erdõs-Renyi bipartite graphs

Without loss of generality we assume that the weights are generic, meaning that not two distinct assignments have the same weight, since this holds with probability 1 in the random model.

A vertex *participates* in an assignment if there is an edge incident to it in the matching. For $0 \leq r \leq k$, we let σ_r be the minimum weight r-matching. Similar to the case of complete bipartite graphs [**31**], we need to prove the following inductive step, which makes the proof follow directly.

$$E[C^d_{k,n,s}] - E[C^d_{k-1,n,s-1}] = \frac{1}{nsd} + \frac{1}{(n-1)sd} + \cdots + \frac{1}{(n-k+1)sd}. \tag{41}$$

We believe that the proofs for the other two conjectures, on the Dispersed-Degree(n, s, d, r) random bipartite graph, should be similar to this proof.

Lemma 5.3. Given an Erdõs-Renyi(n, s, d) bipartite graph. For each vertex $v \in V$ and any subset $U' \subseteq U$, the expected number of neighbors of v in U' is $d \cdot |U'|$.

Proof. Follows from the fact that the sum of $|U'|$ bernoulli variables with parameter d is a binomial variable with parameters $(|U'|, d)$. □

Lemma 5.4. Suppose that $r < \min(n, s)$. Then every vertex that participates in σ_r also participates in σ_{r+1}.

Proof. The proof follows directly from the fact that $H = \sigma_r \triangle \sigma_{r+1}$ (symmetric difference) consists of a single path which is not a cycle. Otherwise we could exchange some edges of σ_r and σ_{r+1}, via a cycle or two paths in H, and find two different matchings of equal size and equal weight, which is not possible since the weights are generic. □

Let us consider a modification of the original bipartite graph where we append one vertex u_{n+1} to U linked to every vertex of V with random exponential weights of rate $\lambda > 0$. The original edges remain exponential with rate 1. In other words, the new graph has vertex sets $\{U' = U \cup \{u_{n+1}\}, V\}$ and edge set $E' = E \cup \{u_{n+1}v \,|\, v \in V\}$.

Lemma 5.5. Condition on the event that u_{n+1} does not participate in σ_r. Then the probability that it participates in σ_{r+1} is:

$$\frac{\lambda}{d(n-r)+\lambda}. \tag{42}$$

Proof. Without loss of generality, the vertices of U participating in σ_r are $\{u_1, \ldots, u_r\}$. From lemma 5.4 we know that only one vertex of $\{u_{r+1}, \ldots, u_{n+1}\}$ can participate in σ_{r+1}. We say that vertex v is assigned to such vertex under σ_{r+1}. Let us define the subset of edges $E_v$to be the neighbors of v that are in $\{u_{r+1}, \ldots, u_{n+1}\}$.

It is easy to see that the vertex assigned to v under σ_{r+1} is the one of E_v that has minimum weight. Therefore, the probability that edge $u_{n+1}v$ is in σ_{r+1} is equal to the probability that edge $u_{n+1}v$ has minimum wight in E_v. Since the weight of the edge $u_{n+1}v$ is exponential with rate λ and all other edges of E_v are exponential with rate 1, then from Lemma 5.3 $|E_v \backslash \{u_{n+1}v\}|$ is in average $d \cdot (n-r)$. Therefore, from the properties of the minimum of exponentially distributed variables, the vertex u_{n+1} will be assigned under σ_{r+1} with probability equal to the one stated in the lemma. □

Corollary 5.6. The probability that u_{n+1} participates in σ_k is:

$$1 - \prod_{i=0}^{k-1} \frac{(n-i)d}{(n-i)d + \lambda} = \lambda \sum_{i=0}^{k-1} \frac{1}{(n-i)d} + O(\lambda^2), \qquad \text{as } \lambda \to 0. \tag{43}$$

Proof. Follows from the fact that the probability that u_{n+1} is in σ_k equals 1 minus the probability that it is not in any σ_r, for $r = 1, \ldots, k$. Which according to lemma 5.5 is:

$$1 - \prod_{i=0}^{k-1} \frac{(n-i)d}{(n-i)d + \lambda} = 1 - \prod_{i=0}^{k-1} \left(1 + \frac{\lambda}{(n-i)d}\right)^{-1} = 1 - \left[\prod_{i=0}^{k-1} \left(1 + \frac{\lambda}{(n-i)d}\right)\right]^{-1}$$

$$= 1 - \left[1 + \lambda \sum_{i=0}^{k-1} \frac{1}{(n-i)d} + O(\lambda^2)\right]^{-1} = \frac{\lambda \sum_{i=0}^{k-1} \frac{1}{(n-i)d} + O(\lambda^2)}{1 + O(\lambda)}$$

This concludes the proof because the denominator in the last expression approaches to one when $\lambda \to 0$. □

Let w denote the weight of the edge $u_{n+1}v_s$. X denote the weight of a minimum k-matching in the graph induced by U and V. And Y denote the weight of a minimum $(k-1)$-matching in the graph induced by U and $V \backslash \{v_s\}$. In other words $X = C_{k,n,s}$ and $Y = C_{k-1,n,s-1}$.

Let I be the indicator variable for the event that $w < X - Y$. That is, the event that the minimum k-matching that contains w is smaller than the minimum k-matching that does not.

Lemma 5.7. When $\lambda \to 0$,

$$E[I] = \left(\frac{1}{nsd} + \frac{1}{(n-1)sd} + \cdots + \frac{1}{(n-k+1)sd}\right)\lambda + O(\lambda^2). \tag{44}$$

Proof. It follows from Corollary 5.6 that the probability that $u_{n+1}v_s$ participates in the minimum k-matching is given by (44). If it does, then $w < X - Y$. Conversely, if $w < X - Y$ and no other edge from u_{n+1} induces a cost smaller than X, then $u_{n+1}v_s$ participates in the minimum k-matching, and when $\lambda \to 0$, the probability that there are two distinct edges from u_{n+1} that induce smaller cost than X is of order $O(\lambda^2)$. □

On the other hand, the fact that w is exponentially distributed of rate λ means that

$$E[I] = P(w < X - Y) = E[1 - e^{-\lambda(X-Y)}] = 1 - E[e^{-\lambda(X-Y)}]. \tag{45}$$

Hence $E[I]$ regarded as a funciton of λ, is essentially the Laplace transform of $X - Y$. In particular, $E[X - Y]$ is the derivative of $E[I]$ at $\lambda = 0$. That is,

$$E[X-Y] = \frac{d}{d\lambda}E[I]|_{\lambda=0} = \frac{1}{nsd} + \frac{1}{(n-1)sd} + \cdots + \frac{1}{(n-k+1)sd}.$$

This finishes the proof of equation 41, establishing conjecture for the Erdõs-Renyi random bipartite graphs with random exponential weights.

Note that the proof is basically based on the fact that the expected number of neighbors of a vertex $v \in V$ from a subset $U' \subseteq U$ is equal to $d \cdot |U'|$. This implies that, if the same is true for the Dispersed-degree random bipartite graphs, then the conjecture in such case is also true according to this proof.

CHAPTER 6

A stochastic assignment problem

1. The problem formulation

Consider the following scenario, which represents a variant of the assignment problem. We have n machines that we need to assign to perform s tasks in a one-to-one relation, at minimum weight. This time we must consider two phases, the assignment phase in the present and the performing phase in the future. It implies that at the point where the tasks must be performed, some of the machines may no longer be available with known probabilities, and some of the tasks may not be needed anymore also with known probabilities. Therefore, if at the performing phase a machine is not performing a task because it was not assigned or because its task is not needed, it incurs a penalization cost for having a machine idle that could have participated in a different schedule. Similar, if at the performing phase a task is not being performed by a machine because it was not assigned or because its assigned machine is not available, it also incurs a penalization cost for having a task unperformed. Note that a machine or a task incur in a penalization cost only if they are present with no mate at the performing phase. Machines and tasks that satisfy this are said to be *idle*. Considering the uncertainty dependence of this problem, the objective is to find an assignment that minimizes the sum of the assignment cost plus the expected value of the penalization costs that we may incur in the resulting assignment.

The *Stochastic Assignment Problem* is modeled as a two-stage stochastic program (Birge and Louveaux 2011 [**35**]). The input to the Stochastic Assignment Problem is the set $\{G, w, p, c\}$. Where G is a bipartite graph $G = (U \sqcup V, E)$ with edge weights $w : E \to \mathbb{Z}$, plus an availability probability $p(u)$ for each $u \in U$ and $p(v)$ for each $v \in V$, and a penalization cost $c(u)$ for each $u \in U$ and $c(v)$ for each $v \in V$. In order to express the problem as a mathematical formulation in the form of a linear program, we need the following auxiliary variables.

- Bernoulli variables $\hat{R}_u = \begin{cases} 1 & \text{with probability } p(u) & : (u \text{ available}), \\ 0 & \text{with probability } (1 - p(u)) & : (u \text{ not available}). \end{cases}$

- Bernoulli variables $\hat{J}_v = \begin{cases} 1 & \text{with probability } p(v) \quad : (v \text{ available}), \\ 0 & \text{with probability } (1 - p(v)) \quad : (v \text{ not available}). \end{cases}$
- Decision variables $x_{uv} = \begin{cases} 1 & \text{if edge } uv \text{ is in the assignment,} \\ 0 & \text{otherwise.} \end{cases}$
- Decision variables $z_u = \begin{cases} 1 & \text{if vertex } u \text{ is idle,} \\ 0 & \text{otherwise.} \end{cases}$
- Decision variables $z_v = \begin{cases} 1 & \text{if vertex } v \text{ is idle,} \\ 0 & \text{otherwise.} \end{cases}$

With the variables already defined, the *Stochastic assignment problem* can be expressed as the following two-phase linear programing model, where $x = \{x_{uv}\}$ is the incidence vector of the matching.

Phase I:

$$\text{minimize:} \quad \sum_{uv \in E} w(uv)x_{uv} + E[Q(x)] \tag{46}$$

$$\text{subject to:} \quad \sum_{v \in N(u)} x_{uv} \leq 1, \qquad \forall u \in U, \tag{47}$$

$$\sum_{u \in N(v)} x_{uv} \leq 1, \qquad \forall v \in V, \tag{48}$$

$$x_{uv} \in \{0, 1\}, \qquad \forall uv \in E. \tag{49}$$

Phase II:

$$Q(x) := \text{minimize:} \quad \sum_{u \in U} c(u)z_u + \sum_{v \in V} c(v)z_v \tag{50}$$

$$\text{subject to:} \quad \sum_{v \in N(u)} \hat{J}_v x_{uv} + z_u \geq \hat{R}_u, \quad \forall u \in U, \tag{51}$$

$$\sum_{u \in N(v)} \hat{R}_u x_{uv} + z_v \geq \hat{J}_v, \quad \forall v \in V, \tag{52}$$

$$z_u, z_v \in \{0, 1\}, \qquad \forall u \in U, v \in V. \tag{53}$$

Note that the resulting matching is not necessarily a perfect matching or even a maximum cardinality matching. It can be a matching of any size, including the empty matching. Also note that in the second phase formulation, the matching induced by x_{uv} is fixed.

2. Reduction to a minimum weight matching problem

Now, the objective is to provide a simpler way to solve this nondeterministic problem. Fortunately we can solve it via a reduction to a regular deterministic minimum weight matching problem, which in turn can be reduced to a minimum weight perfect matching problem as is shown in Appendix A. The reduction starts with the following proposition.

Proposition 6.1. The variables z_u that minimize the second phase model satisfy that:

$$E[z_u] = p(u) \cdot \left(1 - \sum_{v \in N(u)} p(v)x_{uv}\right).$$

Proof. First of all note that in order to achieve the minimization, z_u must be the minimum value that satisfies $z_u \geq \hat{R}_u - \sum_{v \in N(u)} \hat{J}_v x_{uv}$, with $z_u \in \{0,1\}$. Alos note that $\sum_{v \in N(u)} \hat{J}_v x_{uv} \in \{0,1\}$. This lead us to the following three possible cases:

- If $\hat{R}_u = 0$, then $(\hat{R}_u - \sum_v \hat{J}_v x_{uv}) \leq 0$. Therefore $z_u = 0$.
- If $\hat{R}_u = 1$ and $\sum_v \hat{J}_v x_{uv} = 1$, then $(\hat{R}_u - \sum_v \hat{J}_v x_{uv}) = 0$. Therefore $z_u = 0$.
- If $\hat{R}_u = 1$ and $\sum_v \hat{J}_v x_{uv} = 0$, then $(\hat{R}_u - \sum_v \hat{J}_v x_{uv}) = 1$. Therefore $z_u = 1$.

Therefore $z_u = 1$ only in the third case. And since $z_u \in \{0,1\}$, then from the definition of the expected value we have the following, where $P[\cdot]$ means the probability of the event inside the brackets.

$$\begin{aligned} E[z_u] &= 0 \cdot P[z_u = 0] + 1 \cdot P[z_u = 1] \\ &= P[z_u = 1] \\ &= P[\hat{R}_u = 1] \cdot P\left[\sum_v \hat{J}_v x_{uv} = 0\right] \end{aligned}$$

The variable $\hat{R}_u$ is a bernoulli variable with parameter $p(u)$, therefore $P[\hat{R}_u = 1] = p(u)$. Furthermore, since $x_{uv} = 1$ only with the vertex v assigned to u, then $P\left[\sum_v \hat{J}_v x_{uv} = 0\right]$ represents the probability that the vertex v assigned to u will not be available. Note that such probability is one if there is no vertex assigned to u. Using the complement rule, $P\left[\sum_v \hat{J}_v x_{uv} = 0\right] = 1 - P\left[\sum_v \hat{J}_v x_{uv} = 1\right]$, which is the complement of the probability that the vertex assigned to u is available. Using the decision variables x_{uv}, we obtain that $P\left[\sum_v \hat{J}_v x_{uv} = 0\right] = 1 - \sum_v p(v)x_{uv}$, which ends the proof. □

Because of the symmetry of the problem, the following proposition can be proven in exactly the same way as the previous proposition.

Proposition 6.2. The variables z_v that minimize the second phase model satisfy that:

$$E[z_v] = p(v) \cdot \left(1 - \sum_{u \in N(v)} p(u)x_{uv}\right).$$

The two previous propositions help us to get a closed expression for the second phase of our stochastic model. This is formulated and proved in the following proposition.

Proposition 6.3. The expected value of the second phase of the stochastic optimization model $E[Q(x)]$ can be expressed in the closed form:

$$E[Q(x)] = -\sum_{uv\in E} p(u)p(v)[c(u)+c(v)]x_{uv} + \sum_u c(u)p(u) + \sum_v c(v)p(v).$$

Proof. Since the variables z_u, z_v are independent, follows from propositions (6.1) and (6.2) that in the minimization of the second phase we have:

$$\begin{aligned} E[Q(x)] &= E\left[\sum_u c(u)z_u + \sum_v c(v)z_v\right] \\ &= \sum_u c(u)E[z_u] + \sum_v c(v)E[z_v] \\ &= \sum_u c(u)p(u)\cdot\left(1-\sum_{v\in N(u)} p(v)x_{uv}\right) + \sum_v c(v)p(v)\cdot\left(1-\sum_{u\in N(v)} p(u)x_{uv}\right) \\ &= \sum_u c(u)p(u) + \sum_v c(v)p(v) - \sum_{uv\in E} p(u)p(v)c(u)x_{uv} - \sum_{uv\in E} p(u)p(v)c(v)x_{uv}. \end{aligned}$$

Where as we can observe, agrees with the expression in the proposition statement. □

Proposition (6.3) says that given a fixed matching $x = \{x_{uv}\}$, we can compute in closed form the minimum of its expected penalization cost. In the following theorem we see the final step of the reduction from a two-phase nondeterministic linear program to a deterministic linear program. The resulting deterministic formulation is up to a constant factor, a minimum weight matching problem, where the objective is to find a matching of minimum weight regardless of the cardinality, including the empty matching which induces a zero weight.

Theorem 6.4. The two-phase formulation (46 - 53) is equivalent to the following minimum weight matching problem formulation.

(54) min: $$\sum_{uv\in E}[w(uv) - p(u)p(v)(c(u)+c(v))]x_{uv} + \underbrace{\sum_{u\in U} c(u)p(u) + \sum_{v\in V} c(v)p(v)}_{\text{constant}}$$

(55) sub: $$\sum_{v\in N(u)} x_{uv} \le 1, \qquad \forall u \in U,$$

(56) $$\sum_{u\in N(v)} x_{uv} \le 1, \qquad \forall v \in V,$$

(57) $$x_{uv} \in \{0,1\}, \qquad \forall uv \in E.$$

The proof of the theorem follows directly from Proposition 6.3. It is important to make a few remarks about the reduction in Theorem 6.4. First note that although the objective function (54) does no seem to be a minimum weight matching objective function, the term $\sum_u c(u)p(u) + \sum_v c(v)p(v)$ is constant, and only depends on the fixed input of the penalization cost and availability probability vectors $\{c, p\}$. Therefore, during the optimization, we only need to consider the term $\sum_{uv \in E} w'(uv)x_{uv}$, where $w'(uv) := w(uv) - p(u)p(v)(c(u) + c(v))$ is a slight modification of the original input weights. Once we have optimized considering only the non-constant term, we just need to append the constant term value to the resulting optimum value to get the correct optimal value of (54). In other words, we can use the principle: $\min_{x \in D}\{f(x) + C\} = \min_{x \in D}\{f(x)\} + C$.

There are specialized algorithms for solving the minimum weight matching problem. These algorithms have good time complexities, but fail to achieve good performance in practice. In our experience, the best option in practice is to use graph transformations to transform the minimum weight matching problem into a minimum weight one-side perfect matching problem. Then we can transform this minimum weight one-side perfect matching problem into a minimum weight perfect matching problem as is shown in Appendix A. Finally, we can use the Auction algorithm to solve the problem and transform the resulting solution into a solution of the original problem backtracking the transformations. The solution of the original problem turns out to be composed by all the edges of the resulting optimum matching of the transformation that are edges of the original graph.

The results presented in this chapter are part of a work in progress, and has been developed under a collaboration of people from Cinvestav-IPN Marcos C. Vargas and Carlos E. Valencia; from the University of Missouri Haitao Li; from the HP-Labs Cipriano Santos and Iván López; and from the UAM-Azcapotzalco Sergio Pérez; see [**34**].

APPENDIX A

Maximum cardinality weighted matchings

The objective of this section is to provide some useful tools for addressing the problem of finding minimum weight matchings that are not necessarily perfect. This tools consists of a set of graph transformations that lead us to an equivalent minimum weight perfect matching problem, which we already know how to solve efficiently. The equivalence is in the sense that given one optimal solution of the original problem we can directly transform it into an optimal solution of the transformed problem, and viceversa.

Such techniques are useful because in some cases it turns out that the input bipartite graph has no perfect matchings and we are interested in finding the maximum cardinality matching of minimum weight or finding a minimum weight matching regardless of the cardinality. These type of problems can be solved using sophisticated algorithms, but most of them have a difficult implementation and according to our experiments all perform poorly in practice. Therefore the transformations that we will present combined with the great performance of the auction algorithm conform a more efficient alternative.

For this section, the input weighted bipartite graph is $\{G = (U \sqcup V, E), w\}$, with $U = \{u_1, \ldots, u_n\}$, $V = \{v_1, \ldots, v_s\}$, $n \geq s$, $|E| = m$, and $W = \max_{uv \in E} \ |w(uv)|$. Note that the bipartite graph is no longer required to be balanced.

1. The minimum weight maximum cardinality matching problem

Given an integer weighted bipartite graph $\{G, w\}$, this problem consists of finding a maximum cardinality matching of G of minimum weight. We will reduce this problem to the minimum weight perfect matching problem by means of a graph transformation. Such transformation has pros and cons that we will discuss after presenting the transformation.

The concept of the transformation is to make a duplicate of the input weighted bipartite graph, flip it around and then connect the two identical instances by adding one edge between each vertex and its copy at weight $2sW$, like in Figure A.1. More formally, given an integer weighted bipartite graph $\{G = (U \sqcup V, E), w\}$ we consider its flipped copy $G_C = (V' \sqcup U', E')$ preserving the edge weights. We connect these two graphs with the edges $E_U = \{u_i u_i' \mid i = 1, \dots, n\}$ and $E_V = \{v_j v_j' \mid j = 1, \dots, s\}$ at weights $2sW$. Now we have an instance $G_d = (U_d \sqcup V_d, E_d)$, with $U_d = U \sqcup V'$, $V_d = V \sqcup U'$ and $E_d = E \cup E' \cup E_U \cup E_V$. The weight function $w_d : E_d \to \mathbb{Z}$ is then,

$$w_d(e) = \begin{cases} w(u_i v_j) & \text{if } e = u_i v_j \in E, \\ w(u_i v_j) & \text{if } e = v_j' u_i' \in E', \\ 2sW & \text{if } e \in E_U \cup E_V. \end{cases}$$

Note that the new graph is balanced since $|U_d| = |V_d| = n + s$. Also note that it has perfect matchings since $E_U \cup E_V$ is a perfect matching. The important fact is that a minimum weight perfect matching M' of $\{G_d, w_d\}$ induces a minimum weight maximum cardinality matching of $\{G, w\}$ given by $M = M' \cap E$. All this process is illustrated in Figure A.1.

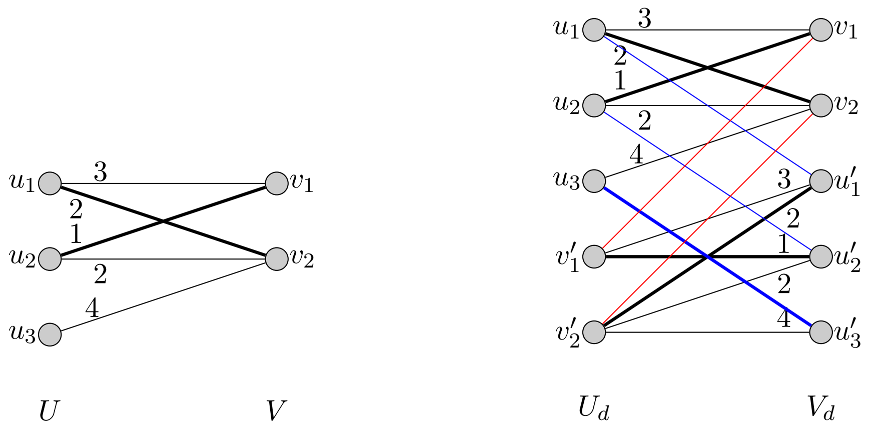


Figure A.1. Reduction from minimum weight maximum cardinality to minimum weight perfect matching. The weight of edges between a vertex and its copy is $2sW$ (blue and red edges).

Proposition A.1. Let $\{G, w\}$ be an integer weighted bipartite graph. If M' is a minimum weight perfect matching of $\{G_d, w_d\}$, then $M = M' \cap E$ is a minimum weight maximum cardinality matching of $\{G, w\}$.

Proof. The matching M is of maximum cardinality because the edges with weights $2sW$ make it preferable to grow the matching as long as possible before adding one edge with such weight.

To prove that M is of minimum weight, consider a diferent maximum cardinality matching M^* of G of smaller weight than M. Then we can form a perfect matching M'' of G_d by combining M^*, its respective copy in G_C and $n + s - 2|M^*|$ edges of $E_U \cup E_V$. Since $|M| = |M^*|$ then,

$$w(M') = (n + s - 2|M|)2sW + 2w(M) > (n + s - 2|M^*|)2sW + 2w(M^*) = w(M'')$$

which is a contradiction to the optimality of M'. □

Now let us discuss the pros and cons. One trivial but important pro is that this transformation always finds a maximum cardinality matching of minimum weight no matter the structure of the instance. A potential problem is that the weight $2sW$ can be very large even in relatively small instances. For example, consider $s = 5000$ and $W = 250000$, then $2sW = 2500000000$, which overflows in a 32-bit integer data type. Therefore unless we have access to the code and have enough RAM space to double the data type precision, we will allways have to make a balance between the size of the instance and the maximum magnitud of the weights.

2. The minimum weight one-side perfect matching problem

A one-side perfect matching in an unbalanced bipartite graph is a matching that covers all the vertices in the smallest side, V in our case. Observe that a one-side perfect matching is a maximum cardinality matching, therefore we can use the previous transformation to find one of minimum weight. But we have invented a more efficient transformation that exploits the structure of this problem.

In summary, the transformation is similar to the previous one, but this time we only connect the biggest side to its copy at any common custom weight, as is shown in Figure A.2. Formally, given an integer weighted bipartite graph $\{G = (U \sqcup V, E), w\}$ we consider its fliped copy $G_C = (V' \sqcup U', E')$ preserving the edge weights. We connect these two graphs with the edges $E_U = \{u_i u_i' \mid i = 1, \ldots, n\}$ all at a common weight $K \in \mathbb{Z}$. We end up with an instance $G_s = (U_s \sqcup V_s, E_s)$, with $U_s = U \sqcup V'$, $V_s = V \sqcup U'$ and $E_s = E \cup E' \cup E_U$. The weight function $w_s : E_s \to \mathbb{Z}$ is then,

$$w_s(e) = \begin{cases} w(u_i v_j) & \text{if } e = u_i v_j \in E, \\ w(u_i v_j) & \text{if } e = v_j' u_i' \in E', \\ K & \text{if } e \in E_U. \end{cases}$$

The new graph is balanced since $|U_s| = |V_s| = n + s$. It has perfect matchings because G has one-side perfect matchings.

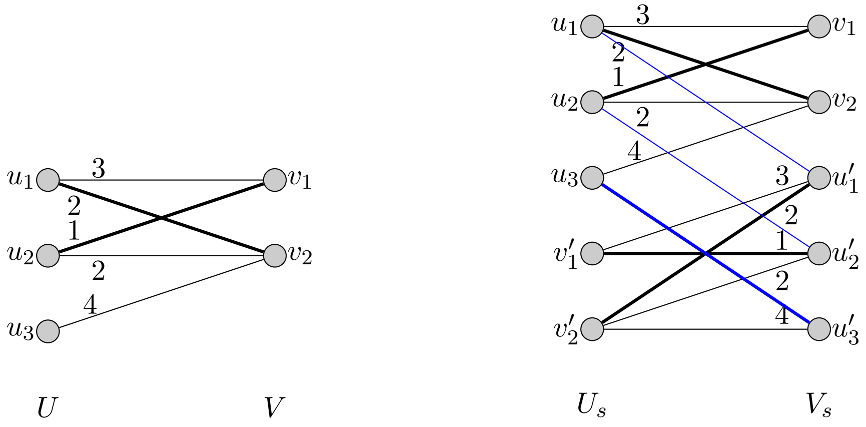


Figure A.2. Reduction from minimum weight one-side perfect matching to minimum weight perfect matching. The weight of edges between a vertex and its copy is $K \in \mathbb{Z}$ (blue edges).

Proposition A.2. Let $\{G, w\}$ be an integer weighted bipartite graph. If M' is a minimum weight perfect matching of the transformation $\{G_s, w_s\}$, then $M = M' \cap E$ is a minimum weight one-side perfect matching of $\{G, w\}$.

Proof. Due to the symmetry of G and its copy, we have that $w(M') = 2w(M) + (n - s)K$ where $(n - s)K$ is constant and therefore minimizing M is equivalent to minimizing M'. □

The only problem with this transformation is that the input instance is forced to have one-side perfect matchings. The big advantage is that now we do not inherit restrictions in the size of the instance or the magnitud of the weights, because the dummy edges E_U can have any common weight of our choice, for example $K = 0$. Another advantage over the previous transformation is that this transformation is solved faster by weight-scaling based algorithms, like the auction algorithm, because a weight $2sW$ imply a bigger number of scaling phases.

Note that this transformation as well as the previous one, needs to double the size of the input instance. One reasonable question is if we can do better when the input graph is almost balanced, for example $n = s+1$ or $n-s \approx \log(n)$. It turns out that we can via a another transformation. The idea of this transformation is to append a set V' of $(n-s)$ artificial vertices to V such that $|V \cup V'| = |U| = n$. And then append all possible edges between U and V' at a common weight $K \in \mathbb{Z}$. This is shown in Figure A.3. Formally, given an integer weighted bipartite graph $\{G = (U \sqcup V, E), w\}$ we define a set of artificial vertices $V' = \{v_{s+1}, \ldots, v_n\}$ and a set of edges $E' = \{u_i v_j \mid u_i \in U,\ v_j \in V'\}$ all with weight $K \in \mathbb{Z}$. Then we have an instance $G_a = (U \sqcup V_a, E_a)$, with $V_a = V \sqcup V'$ and $E_a = E \cup E'$. The weight function $w_a : E_a \to \mathbb{Z}$ is then,

$$w_a(e) = \begin{cases} w(u_i v_j) & \text{if } e = u_i v_j \in E, \\ K & \text{if } e = u_i v_j \in E'. \end{cases}$$

The new graph is balanced since $|U| = |V_a| = n$. It has perfect matchings because G has one-side perfect matchings.

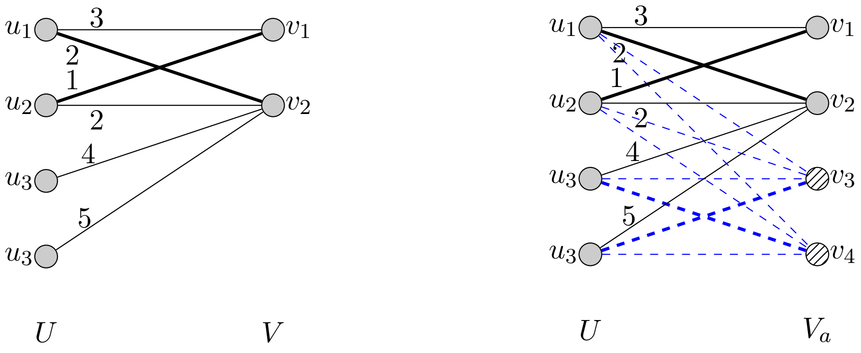


Figure A.3. Reduction from minimum weight one-side perfect matching to minimum weight perfect matching, suited for almost balanced graphs. The weight of the edges not in E is $K \in \mathbb{Z}$ (blue dashed edges).

Proposition A.3. Let $\{G, w\}$ be an integer weighted bipartite graph. If M' is a minimum weight perfect matching of the transformation $\{G_a, w_a\}$, then $M = M' \cap E$ is a minimum weight one-side perfect matching of $\{G, w\}$.

Proof. We have that $w(M') = w(M) + (n - s)K$ where $(n - s)K$ is constant and therefore minimizing M is equivalent to minimizing M'. □

In this case, the transformation appends $(n-s)n$ edges, which is quadratic if $n-s = O(n)$ implying a remarkable poor performance. But if $n - s$ is small enough, then then this transformation will have much better performance than the previous transformation G_s. Note that since the auxiliary edges form a complete bipartite graph all with the same weight, then in practice it is not necessary to store them explicitly. This means that we can implement this transformation with no additional space other than the occupied by the original input graph.

3. The minimum weight matching problem

The minimum weight matching problem consists of finding a matching of minimum weight regardless of the cardinality, including the zero weight empty matching. The input bipartite graph can be either balanced or unbalanced. In this case we will not reduce the problem directly to the minimum weight perfect matching problem, instead we will reduce it to the minimum weight one-side perfect matching problem, which we already know how to solve.

The idea of this transformation is to make a copy of the small side V and connect each vertex to its copy at zero weight. See Figure A.4. That is, given an integer weighted bipartite graph $\{G = (U \sqcup V, E), w\}$, make a copy $V' = \{v'_1, \ldots, v'_s\}$ of V and connect it with the edges $E_V = \{v'_j v_j \mid j = 1, \ldots, s\}$ all at zero weight. Thus we obtain an unbalanced instance $G_m = (U_m \sqcup V, E_m)$, with $U_m = U \sqcup V'$ and $E_m = E \cup E_V$. The weight function $w_m : E_m \to \mathbb{Z}$ is then,

$$w_m(e) = \begin{cases} w(u_i v_j) & \text{if } e = u_i v_j \in E, \\ 0 & \text{if } e = v'_j v_j \in E_V. \end{cases}$$

The new graph is unbalanced but has one-side perfect matchings because E_V is one such matching.

Proposition A.4. Let $\{G, w\}$ be an integer weighted bipartite graph. If M' is a minimum weight one-side perfect matching of the transformation $\{G_m, w_m\}$, then $M = M' \cap E$ is a minimum weight matching of $\{G, w\}$.

Proof. We have that $w(M') = w(M)$ therefore minimizing M is equivalent to minimizing M'. □

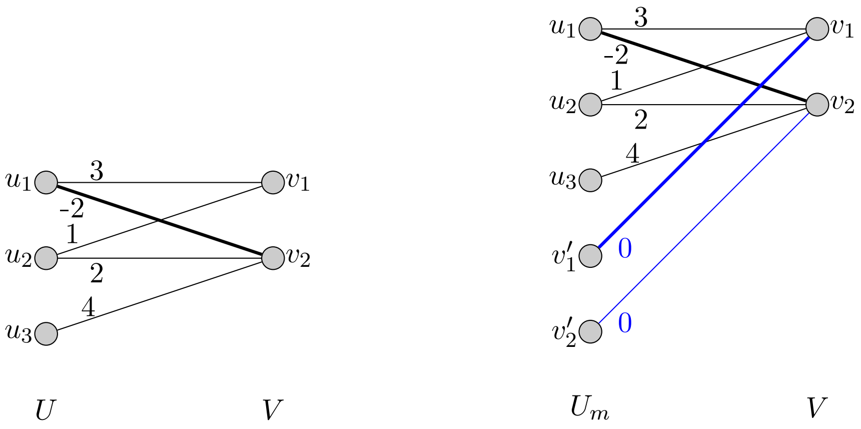


Figure A.4. Reduction from minimum weight matching problem to a minimum weight one-side perfect matching problem.

Bibliography

[1] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment problems*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2009.

[2] L. Lovász and M.D. Plummer, *Matching theory*, AMS Chelsea Publishing, Providence, RI, 2009.

[3] D. Bertsimas and J.N. Tsitsiklis, *Introduction to linear optimization*, Athena Scientific, Belmont, MA, 1997.

[4] A. Schrijver, *Combinatorial optimization: Polyhedra and efficiency*, Algorithms and Combinatorics 24, Springer-Verlag, Berlin, 2003.

[5] J. A. Bondy and U.S.R. Murty, *Graph Theory*, Graduate Texts in Mathematics 244, Springer-Verlag, New York, 2008.

[6] R. Diestel, *Graph Theory*, Graduate Texts in Mathematics 173, Springer-Verlag, Heidelberg, 2010.

[7] H.W. Kuhn, *The hungarian method for the assignment problem*, Naval Research Logistics Quarterly 2, Pages 83–97, 1955.

[8] H.N. Gabow, *Scaling algorithms for network problems*, Journal of computer and system sciences 31, Pages 148–168, 1985.

[9] E.A. Dinic and M.A. Kronrod, *An algorithm for the solution of the assignment problem*, Soviet Math. Dokl. Vol 10, Pages 1324–1326, 1969.

[10] D.P. Bertsekas, *Auction algorithms for network flow problems: A tutorial introduction*, Computational optimization and applications 1, Pages 7–66, 1992.

[11] H.N. Gabow and R.E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Computation Vol. 18, No. 5, Pages 1013–1036, 1989.

[12] A.V. Goldberg and R. Kennedy, *An efficient cost scaling algorithm for the assignment problem*, Mathematical programming 71, Pages 153–177, 1995.

[13] J. Hopcroft and R.M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput. 2, Pages 225–231, 1973.

[14] H. Halt, N. Blum, K. Mehlhorn and M. Paul, *Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/log\, n})$*, Information Processing Letters 37, Pages 237–240, 1991.

[15] F. Glover, *Maximum matching in a convex bipartite graph*, Naval Research Logistics Quarterly, Vol. 14, Pages 313–316, 1967.

[16] K.S. Booth and G.S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms*, Journal of Computer and System Sciences, Vol. 13, Pages 335–379, 1976.

[17] H.N. Gabow and R.E. Tarjan, *A linear-time algorithm for a special case of disjoint set union*, Journal of Computer and System Sciences, Vol. 30, Pages 209–221, 1985.

[18] D. Coppersmith and S.Winograd, *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation, Vol. 9, Pages 251–280, 1990.

[19] L. Lovász, *On determinants, matchings and random algorithms*, Fundamentals of Computation Theory FCT'79, Vol. 2, pages 565–574, Akademie Verlag, Berlin, 1979.

[20] T.Uno, *A Fast Algorithm for Enumerating Bipartite Perfect Matchings*, Lecture Notes in Computer Science, Springer Verlag, Vol. 2223, Pages 367–379, 2001.

[21] K. Fukuda and T. Matsui, *Finding All the Perfect Matchings in Bipartite Graphs*, Appl. Math. Lett, Vol. 7, No. 1, Pages 15–18, 1994.

[22] T. Tassa, *Finding All Maximally-Matchable Edges in a Bipartite Graph*, Theoretical Computer Science 423, Pages 50–58, 2012.

[23] M.C. Costa, *Persistency in maximum cardinality bipartite matchings*, Operation Research Letters 15, Pages 143–149, 1994.

[24] R. Tarjan. *Depth first search and linear graph algorithms*, SIAM Journal on Computing, Vol. 1, Pages 146–160, 1972

[25] J.C. Régin, *A filtering algorithm for constraints of difference in CSPs*, Proceedings of the 12th National Conference on Artificial Intelligence (AAAI), Pages 362–367, 1994.

[26] I.P. Gent, I. Miguel and P. Nightingale, *Generalised arc consistency for the AllDifferent constraint: An empirical survey*, Artificial Intelligence 172, Pages 1973–2000, 2008.

[27] A. Pothen and C.J. Fan, *Computing the Block Triangular Form of a Sparse Matrix*, ACM Transactions on Mathematical Software, Vol. 16 Issue 4, Pages 303–324, 1990.

[28] J. Cheriyan, *Randomized $O(M|V|)$ algorithms for problems in matching theory*, SIAM J. Comput. 26, Pages 1635–1669, 1997.

[29] M. O. Rabin and V. V. Vazarani, *Maximum matchings in general graphs through randomization*, J. Algorithms 10, Pages 557–567, 1989.

[30] P.A. Krokhmal and P.M. Pardalos, Random assignment problems, European J. Oper. Res. 194 (2009), no. 1, 1–17.

[31] Johan Wästlund, *An easy proof of the $\zeta(2)$ limit in the random assignment problem*, Electronic Communications in Probability 14 (2009), p. 261-269.

[32] C. E. Valencia and M. C. Vargas, Optimum matchings in a weighted bipartite graph, Boletín de la Sociedad Matematica Mexicana **22** 1 (2016), 1-12.

[33] C. E. Valencia and M. C. Vargas, The random assignment problem on Erdõs-Renyi bipartite random graph, in progress.

[34] Hitao Li, Marcos C. Vargas, Cipriano Santos, Lyle Ramshaw, Ivan Lopez, Sergio Perez, and Carlos Valencia, Optimizing Large-Scale Stochastic Resource Planning, in progress.

[35] Birge, J. R. and F. Louveaux. Introduction to Stochastic Programming. New York, Springer, 2011.

Index

Marcos César Vargas Magaña