

Chapter 3

Matching

A.M.H. Gerards

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

1. Introduction

Matching is pairing: dividing a collection of objects into pairs. Typically the objective is to maximize total profit (or minimize cost), where the profit of each possible pair is known in advance.

For a more formal definition, let G be an undirected graph with node set V and edge set E . A subset M of E such that no two edges in M are incident to a common node is a *matching*. If M has exactly one edge incident to each node $v \in V$, we call M a *perfect matching*. The *maximum weight matching problem* with respect to the *weights* w on the edges is:

Find a matching M with total weight $\sum_{e \in M} w_e$ as large as possible. (1)

The *minimum weight perfect matching problem* is:

Find a perfect matching M with total weight $\sum_{e \in M} w_e$ as small as possible. (2)

A matching problem is defined by two parameters: the graph G and the weights w . We classify matching problems by these different parameters:

The weights – cardinality or weighted matching. Finding a maximum cardinality matching, i.e. considering all edges to have weight one, is easier than dealing with more general weights. Moreover, an algorithm for finding a maximum cardinality matching can be, and in our presentation is, used as a subroutine for solving the general weighted problem. Therefore, we discuss the cardinality case first, in Section 2, and study general weights later, in Section 6.

The graph – bipartite or non-bipartite matching. Matching problems are significantly easier in bipartite graphs. So, we discuss several topics in bipartite matching before venturing into the complications of the non-bipartite case.

Matching theory is one of the cornerstones of mathematical programming. Yet, matchings are not as ubiquitous in practice as network flow problems (for applications of network flows, see Ahuja, Magnanti & Orlin [1989]) and, when they do arise in practice, it is most often in bipartite graphs where they can be

modeled as network flow problems anyway (see Section 3.1). So, to what does 'Matching' owe its prominence?

First, there is the position of matching problems between the 'easier' problems like network flows, and the hard (NP-hard) problems like general integer linear programming. This position has been pointed out by Edmonds & Johnson [1970]. It is probably best expressed by the qualification: "*Optimum matching problems ... constitute the only class of genuine integer programs for which a good solution is known*" [Cunningham & Marsh, 1978]. In a footnote, Cunningham and Marsh clarify this statement with: "... *Every other class of well-solved combinatorial problems is not 'genuine' because either: (a) No explicit formulation as an integer program using a reasonable amount of data is known (example: minimum spanning tree problems); or (b) When such a formulation is known, the resulting linear program already has integer-valued optimal solutions (example: network flow problems).*"

Second, and perhaps more intrinsic to the importance of matching, there is the intricate structure of matching theory. Just a glimpse in the excellent book *Matching Theory* by Lovász & Plummer [1986] should convince the reader of this. For instance, there are the structural descriptions of the class of all maximum cardinality matchings in a graph. In this chapter we see how one of these, the Edmonds–Gallai structure (see Section 4.1), helps in understanding an algorithm for finding a minimum weight perfect matching (see Section 6.2). Third, there is the 'self-refining' property of matching theory: it contains a wide class of its generalizations as special cases (see Sections 3 and 7).

1.1. Examples of matching problems

We begin with four examples of matching problems. A classic example is:

The assignment problem. Suppose n tasks are to be carried out and each must be assigned to a single person. We have a staff of n people available and each person can be assigned only one task. Moreover, we know for each person p and task t a number $w_{p,t}$ quantifying the productivity of p when carrying out t . Now, the question is: How do we assign the tasks to the people, i.e. make task-person pairs, so that the total productivity is as large as possible?

Clearly, this is a bipartite matching problem. An example of a non-bipartite matching problem is:

The oil well drilling problem [Devine, 1973]. Suppose we are given the locations of oil deposits. We want to exploit all the deposits at minimum drilling cost. It is technically feasible to access two deposits with one well: drill a hole to the first deposit, and then continue drilling from that same hole, possibly at a different angle, to the second deposit. The oil is then brought up from both deposits via concentric pipes. We know the savings possible from combined drilling operations for each pair of deposits. The question is: How do we combine the drilling operations so that the savings are as large as possible?