Efficient Algorithms for Finding Maximum Matching in Graphs

ZVI GALIL

Department of Computer Science, Columbia University, New York, N.Y., 10027 and Tel-Aviv University, Tel-Aviv, Israel

This paper surveys the techniques used for designing the most efficient algorithms for finding a maximum cardinality or weighted matching in (general or bipartite) graphs. It also lists some open problems concerning possible improvements in existing algorithms and the existence of fast parallel algorithms for these problems.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Algorithmic tools, the asexual case, assignment problem, augmenting path, blossoms, data structures, d-heap, duality, ET, generalized priority queue, Main Theorem of Botany, matching, monsters, moonlighting, polygamy, primal–dual method, shmathematics, shrink, warm-up

INTRODUCTION

There are no recipes for designing efficient algorithms. This is somewhat unfortunate from the point of view of applications: Any time we have to design an algorithm, we may have to start (almost) from scratch. However, it is fortunate from the point of view of researchers: It is unlikely that we are going to run out of problems or challenges.

Given a problem, we want to find an algorithm that solves it efficiently. There are three stages in designing such algorithms:

(a) *Shmathematics.* Initially, we use some simple mathematical arguments to characterize the solution. This leads to a simple algorithm that is usually not very efficient.

(b) *Algorithmic Tools.* Next, we try to apply a number of algorithmic tools to speed up the algorithm. Examples of such tools are "divide and conquer" and dynamic programming [Aho et al. 1974]. Alternatively, we may try to find a way to reduce the number of steps in the original algorithm by finding a better way to organize the information.

(c) *Data Structures.* Sometimes we can speed up an algorithm by using an efficient data structure that supports the primitive operations used by the algorithm. We may even resort to the introduction of monsters: very complicated data structures that bring about some asymptotic speedup that is usually meaningful only for very large problem size. (For a real-life monster see Galil [1980].)

In these three stages we sometimes use a known technique: a certain result in mathematics, say, or a known algorithmic tool or data structure. In the more interesting problems we need to invent new techniques