CMSC 651 Advanced Algorithms	Lecture 4
Lecturer: Samir Khuller	Th. Feb 7. 2002

Notes by Samir Khuller.

13 Two Processor Scheduling

We will also talk about an application of matching, namely the problem of scheduling on two processors. The original paper is by Fujii, Kasami and Ninomiya [7].

There are two identical processors, and a collection of n jobs that need to be scheduled. Each job requires unit time. There is a precedence graph associated with the jobs (also called the DAG). If there is an edge from i to j then i must be finished before j is started by either processor. How should the jobs be scheduled on the two processors so that all the jobs are completed as quickly as possible. The jobs could represent courses that need to be taken (courses have prerequisites) and if we are taking at most two courses in each semester, the question really is: how quickly can we graduate?

This is a good time to note that even though the two processor scheduling problem can be solved in polynomial time, the three processor scheduling problem is not known to be solvable in polynomial time, or known to be NP-complete. In fact, the complexity of the k processor scheduling problem when k is *fixed* is not known.

From the acyclic graph G we can construct a *compatibility* graph G^* as follows. G^* has the same nodes as G, and there is an (undirected) edge (i, j) if there is no directed path from i to j, or from j to i in G. In other words, i and j are jobs that can be done together.

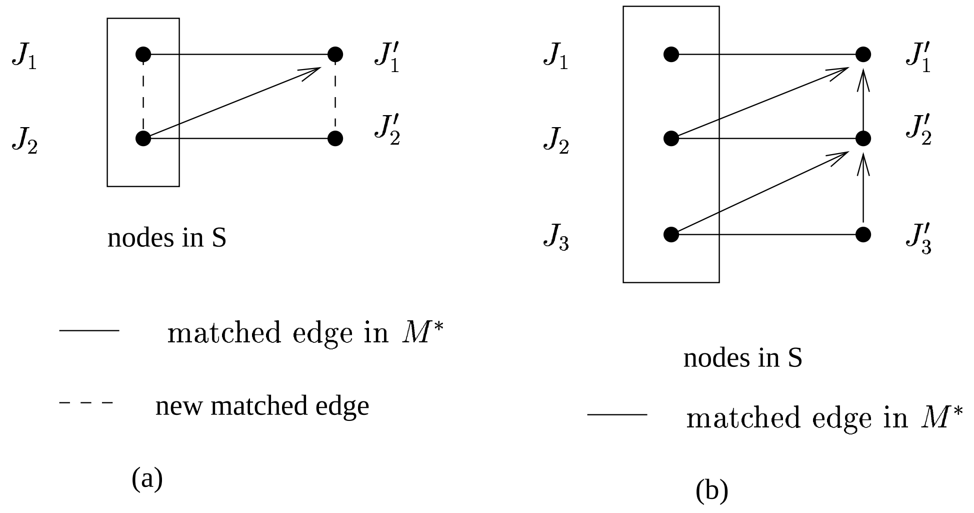

Figure 12: (a) Changing the schedule (b) Continuing the proof.

A maximum matching in G^* is the indicates the maximum number of pairs of jobs that can be processed simultaneously. Clearly, a solution to the scheduling problem can be used to obtain a matching in G^*. More interestingly, a solution to the matching problem can be used to obtain a solution to the scheduling problem!

Suppose we find a maximum matching in M in G^*. An unmatched vertex is executed on a single processor while the other processor is idle. If the maximum matching has size m^*, then $2m^*$ vertices are matched, and $n - 2m^*$ vertices are left unmatched. The time to finish all the jobs will be $m^* + n - 2m^* = n - m^*$. Hence a maximum matching will minimize the time required to schedule all the jobs.

We now argue that given a maximum matching M^*, we can extract a schedule of size $n - m^*$. The key idea is that each matched job is scheduled in a slot together with another job (which may be different from the job it was matched to). This ensures that we do not use more than $n - m^*$ slots.

Let S be the subset of vertices that have indegree 0. The following cases show how a schedule can be constructed from G and M^*. Basically, we have to make sure that for all jobs that are paired up in M^*, we pair them up in the schedule. The following rules can be applied repeatedly.

1. If there is an unmatched vertex in S, schedule it and remove it from G.
2. If there is a pair of jobs in S that are matched in M^*, then we schedule the pair and delete both the jobs from G.

If none of the above rules are applicable, then all jobs in S are matched; moreover each job in S is matched with a job not in S.

Let $J_1 \in S$ be matched to $J_1' \notin S$. Let J_2 be a job in S that has a path to J_1'. Let J_2' be the mate of J_2. If there is path from J_1' to J_2', then J_2 and J_2' could not be matched to each other in G^* (this cannot be an edge in the compatibility graph). If there is no path from J_2' to J_1' then we can *change* the matching to match (J_1, J_2) and (J_1', J_2') since (J_1', J_2') is an edge in G^* (see Fig. 12 (a)).

The only remaining case is when J_2' has a path to J_1'. Recall that J_2 also has a path to J_1'. We repeat the above argument considering J_2' in place of J_1'. This will yield a new pair (J_3, J_3') etc (see Fig. 12 (b)). Since the graph G has no cycles at each step we will generate a distinct pair of jobs; at some point of time this process will stop (since the graph is finite). At that time we will find a pair of jobs in S we can schedule together.