On the connection between matching theory and switching

Konstantinos Psounis
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
kpsounis@leland.stanford.edu

December 8th 2000

1 Introduction

Matching theory is a traditional topic of Combinatorics with many applications. Recently, concepts of matching and in particular of bipartite matching have been used to analyze and solve problems related to the design of scheduling algorithms for high speed network switches.

The purpose of this work is to present the main results of switching, and highlight how matching theory is used to analyze such problems. In the next section, we briefly present some results of Matching Theory [8] and the Stable Marriage Problem [7]. Also, we summarize the basic results of switching. In Section 3 we present in more detail various results of switching and show how matching theory is used in the context of such problems.

2 Background

2.1 Matching

Given a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, a maximum matching for $G$ is the largest set $M^* \subseteq E$ such that no two edges in $M^*$ have a vertex in common.

A graph is bipartite if its set of vertices $V$ can be partitioned into two sets $V_L$ and $V_R$ such that every edge in $E$ has one endpoint in $V_L$ and one in $V_R$. A maximum matching for a bipartite graph is defined as before. In the following discussion we restrict ourselves to bipartite graphs only. A maximal
matching is one where each vertex in \( V_l \) is either part of the matching, or all the vertices in \( V_r \) that it can possibly connect to are already matched to other vertices. One can easily see that every maximal matching is at least half the size of a maximum matching.

Associate with each edge \( e \) a weight \( w_e \). Then \( M^*_w \) is a maximum weight matching if it maximizes \( \sum_{e \in M} w_e \) over all possible matchings \( M \). Note that the maximum weight matching may not be a maximum matching. Let the cardinality of both \( V_l \) and \( V_r \) be \( n \). Then, the most efficient algorithm for solving the problem of maximum matching converges in \( O(n^{5/2}) \) time [5], and the most efficient algorithm for solving the problem of maximum weight matching converges in \( O(n^3) \) \(^1\) running time [10].

We now turn into the stable marriage problem. Let \( M \) and \( W \) be two finite sets of \( n \) men and women respectively. A matching is a set of monogamous marriages between the men and the women. Suppose that each man has an order of preference for the women and each woman an order of preference for the men. A matching is \textit{unstable} if a man \( A \) and a woman \( a \), not married to each other, mutually prefer each other to their spouses. A matching that is not unstable is called a \textit{stable marriage}. It is known that there exists at least one stable marriage solution in this problem [7].

Suppose we seek stable solutions with the extra condition that each person be married to someone appearing on his or her possibly incomplete list of preferences. Then, the existence theorem of stable matching for complete lists does not generalize to the case of incomplete lists as it can be easily seen by a counter example. For example, the only possible matching in the situation below is \((Aa, Bb, Cc)\) and this is clearly unstable because of \( B \) and \( c \).

\[
\begin{align*}
A: & a & a: & C & A & B \\
B: & c & a & b & b: & B & A & C \\
C: & c & a & c: & A & B & C \\
\end{align*}
\]

If we drop the requirement that all men and women get married, then maximal solutions are also possible. For example, if \( A \) does not prefer getting married to \( b \) or \( c \) from staying single, then \((Bc, Ca)\) is a stable maximal solution. Let preference lists have the following property: If woman \( x \) is not in the preference list of man \( Y \), then man \( Y \) is not in the preference list of woman \( x \). For example, the preference lists above would look like

\[
\begin{align*}
A: & a & a: & C & A & B \\
B: & c & a & b & b: & B \\
C: & c & a & c: & B & C \\
\end{align*}
\]

\(^1\)\(O(mn \log_{\log n} n), m \leq n^2\)
Then, \( (Bc, Ca) \) is a stable maximal solution irrespectively of whether \( A \) prefers to get married to \( c \) from staying single.

A stable matching can be found by the Gale-Shapley algorithm [4]. The worst case complexity of the algorithm is \( O(n^2) \) \(^2\) while on average its complexity is \( O(n \log n) \) [7].

2.2 Switching

Conceptually, a space-division packet switch, or simply a switch, is a box with \( n \) inputs and \( n \) outputs that routes (switches) the packets arriving on its inputs to the appropriate outputs.

Many commercial switches today employ output-queueing (OQ) (figure 1). When a packet arrives at an OQ switch, it is immediately placed in a queue that is dedicated to its outgoing line, where it will wait until departing from the switch. Thus, OQ switches have \( n \) queues, one at each output. Output queued switches are known to maximize throughput and they are able to bound the delay of packets, but they are impractical for switches with high line rates and/or a large number of ports. The reason is that the fabric and memory of an \( n \times n \) switch must run \( n \) times as fast as the line rate.

On the other hand, the fabric and the memory of an input-queued (IQ) switch (figure 1) need only to run as fast as the line rate. When a packet arrives to an IQ switch, it is placed in a queue that is dedicated to its input line, where it waits until being scheduled to depart from the switch. Thus, IQ switch have \( n \) queues, one at each input. The main problem of an input queue switch with FIFO queues is head-of-line blocking, which can limit the throughput to just 58.6\% [6].

Head-of-line blocking refers to the situation where an input queue \( i \) has a packet destined to some available output \( j \), but the packet cannot be sent to \( j \) because there is another packet sitting in front of it that is destined to an unavailable output \( k \). For example, in the IQ switch of figure 1 let packets 1 and 2 be destined to output \( a \) and packet 3 to output \( b \). If 1 is scheduled to be serviced, 3 is head-of-line blocked. Indeed, output \( b \) is available but 3 cannot be serviced because of 2 that is sitting in front of 3 and is destined to the unavailable output \( a \).

To eliminate head of line blocking, each input should have \( n \) separate FIFO queues, one for each output. We call this design a virtual output queued (VOQ) switch. Notice that in the switch with VOQs in figure 1

\(^2\)In Gale-Shapley algorithm men make proposals to women, and the maximum number of proposals required to find a stable marriage is \( O(n^2) \).
packet 3 is not head-of-line blocked now. Such designs can achieve 100% throughput like output queued switches. One queue per input suffices to achieve 100% throughput only if it is possible to service any packet of the queue, irrespectively of its position.

In practice, we are not only interested in the throughput of a switch, but also in the latency of individual packets. The method to reduce latency is speedup. A switch with a speedup of $S$ can remove up to $S$ packets from each input and deliver up to $S$ packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Input queued switches have a speedup of 1, while output queued switches have a speedup of $n$. For values of $S$ between 1 and $n$, packets need to be buffered at the inputs before switching as well as at the outputs after switching. We call this architecture a combined input and output queued (CIOQ) switch. Figure 1 schematically presents the architectures mentioned above.

Since OQ switches can both achieve high throughput and bound delay, we aim to have a design with the characteristics of OQ switches, without the
need for a speedup of \( n \). Using a CIOQ switch, with virtual output FIFO queues at the input, and a properly designed scheduling algorithm, it has been shown that a speedup of 4 suffices to mimic the behavior of an output queued switch [9]. A more recent work showed that the speedup required to mimic the behavior of an output queued switch is even less, namely \( 2 - 1/n \) [3].

A scheduling algorithm dictates which inputs will send packets to which outputs at each time slot. It is easy to see that the process of finding input-output pairs, can be modeled as a bipartite matching problem.

3 Solving switching problems using matching techniques

3.1 Achieving 100\% throughput using maximum weight matching

Let \((V_l, V_r)\) be a bipartite graph associated with a switch such that the elements of \(V_l\) and \(V_r\) are the inputs and outputs of the switch respectively. Then, one can use the maximum matching algorithm to decide which packets will be transferred to the outputs per time slot, by matching inputs and outputs. Such a strategy attempts to maximize the number of connections per time slot, and hence to maximize the instantaneous allocation of bandwidth. This technique has been used by many scheduling algorithms in the past [1].

However, there are two problems with that approach. First, under admissible traffic\(^3\) it can lead to instability [2]. Second, it is easy to see that under inadmissible traffic, it can lead to starvation\(^4\). Indeed, consider two inputs \(A, B\) and two outputs \(a, b\). Assume input \(B\) sends traffic to output \(a\) at the maximum possible rate, and input \(A\) has traffic for both outputs \(a\) and \(b\). The maximum matching algorithm will always choose the "cross" traffic, resulting into starving the connection from input \(A\) to output \(a\).

The maximum matching algorithm knows only whether an input queue is empty or non-empty. Therefore, if the occupancies of some queues begin to increase, this algorithm does not know to favor those queues and reduce their backlog. On the other hand, a maximum weight algorithm can take into account the occupancy of each input queue or the waiting time of a packet at the head-of-line of a queue. Using such a quantity as the weight

\(^3\)Admissible is the traffic that is less or equal to the capacity of the switch.

\(^4\)An input-output pair isstarved if it is never chosen in the matching.
of each possible pair, it can be shown\textsuperscript{5} that a maximum weight matching algorithm with a design employing virtual output queues can achieve 100% throughput under any traffic pattern [2].

3.2 Bounding packet delay using stable marriage

Even though a combination of virtual output queuing and maximum weight matching guarantees 100% throughput, it cannot guarantee perfect mimicking of an OQ switch. Thus, such a design cannot bound delay and provide quality of service.

A CIOQ system is said to behave identically to an OQ switch if, under identical inputs, the departure time of every packet from both switches is identical. The challenge is to come up with a scheduling algorithm that will require a reasonably small speedup for perfect mimicking to take place.

Such an algorithm is the so called \textit{Most Urgent Cell First Algorithm (MUCFA)}\textsuperscript{[9]}\textsuperscript{6}. The algorithm seeks a stable marriage between inputs and outputs. To describe the preference lists of inputs and outputs that are used in the process of coming up with a stable marriage, we need to introduce some definitions and briefly describe the basics of the algorithm.

For a switch with speedup \( S \), a time slot is said to be divided into \( S \) equal phases. During each phase, the switch can remove at most one packet from each input and transfer at most one packet to each output. Associate with the CIOQ switch an OQ switch and impose identical inputs to both of them. Each packet in the CIOQ switch corresponds to a packet in the OQ switch. We call the later the \textit{clone} of the former. The \textit{urgency} of a packet in a CIOQ switch at any time is the distance its clone is from the head of the output buffer in the corresponding reference OQ switch.

A phase by phase description of MUCFA follows:

1. At the beginning of each phase outputs try to obtain their most urgent packets from the inputs\textsuperscript{6}.

2. If more than one output requests an input, then the input will grant to that output whose packet has the smallest urgency number. If there is a tie, the output with the smallest port number wins (any policy for ties will do).

\textsuperscript{5}The proof uses a simple linear programming argument and quadratic Lyapunov functions

\textsuperscript{6}Thus, outputs make the proposals to inputs
3. Outputs that lose contention at an input will try to obtain their next most urgent packet from another input that is not matched with its final output pair.

4. When no more matching of inputs and outputs is possible, packets are transferred and MUCFA goes to the next phase.

Let $VOQ_{ij}$ denote the virtual output queue at input $i$ that queues packets destined at output $j$. Output $j$ assigns a preference value to each input $i$, equal to the urgency of the packet at the head-of-line of $VOQ_{ij}$. The preference list of output $j$ is the ordered set of its preference values for each input. Likewise, each input assigns a preference value for each output, and creates the preference list accordingly.

In figure 2 we give an example of a CIOQ switch with three inputs and three outputs, employing virtual output queues. Packets are denoted by their urgencies. The preference lists in the current phase are as follows:

- $A$: $a$ $b$ $c$  
- $B$: $c$ $b$  
- $C$: $c$ $a$

Notice that preference lists are incomplete. However, if output $x$ is not in input’s $Y$ preference list, then input $Y$ is not in outputs $x$ preference lists. Due to that property one can see that there always exists a stable maximal
(not necessarily complete) matching. Alternatively, if $VQ_{ij}$ is empty we could assign a preference value of $+\infty$ to input $i$ for output $j$ and to output $j$ for input $i$, and solve the complete list stable marriage problem that is known to always have a solution.

A proof involving some combinatorial arguments is given in [9] that a speedup of 4 suffices for that system to mimic an OQ switch. Also, due to the manner in which preference lists are drawn, the worst-case number of iterations required in the switching problem is $n[9]$. This is due to the fact that the Gale-Shapley algorithm can be implemented in parallel; all outputs try to obtain an input at the same time and at least one input-output pair will be fixed per iteration. Notice that the total number of proposals is still $O(n^2)$.

Building upon the speedup result above, the authors in [3] demonstrate that a CIOQ switch with $n$ inputs and $n$ outputs and a speedup of only $2 - 1/n$, can provide precise emulation of an OQ switch. Finally, in addition to the sufficiency of the speedup of $2 - 1/n$, the authors demonstrate its necessity by a counter-example.

References


